SPIS TREŚCI

WYKAZ NAJWAŻNIEJSZYCH SKRÓTÓW I OZNACZEŃ

CAD	**C**omputer **A**ided **D**esign Komputerowe wspomaganie projektowania
CAM	**C**omputer **A**ided **M**anufacturing Komputerowe wspomaganie wytwarzania
CNC	**C**omputerized **N**umerical **C**ontrol Komputerowe sterowanie urządzeń numerycznych
DXF	**D**ata E**x**change **F**ormat Format pliku służący wymianie danych wektorowych np. w systemach CAD
HTML	**H**yper**T**ext **M**arkup **L**anguage Hipertekstowy język znaczników, wykorzystywany do tworzenia stron WWW
KRL	**K**UKA **R**obot **L**anguage Język programowania robotów KUKA
KSS	**K**UKA **S**ystem **S**oftware System programowania robota KUKA
OpenGL	**O**pen **G**raphics **L**ibrary Otwarte biblioteka graficzna, stosowana do tworzenia grafiki komputerowej
PC	**P**ersonal **C**omputer Komputer osobisty
PCB	**P**rinted **C**ircuit **B**oard Obwód drukowany - płyta do montażu podzespołów elektronicznych
SMD	**S**urface **M**ounted **D**evices Elementy elektroniczne stosowane do montażu powierzchniowego
SMT	**S**urface **M**ount **T**echnology Technologia powierzchniowego montażu elementów elektronicznych
TCP	**T**ool **C**enter **P**oint Punkt środkowy narzędzia zamocowane na robocie przemysłowym
XML	**E**xtensible **M**arkup **L**anguage Język znaczników stosowany do zapisu danych strukturalnych

WSTĘP

Niniejsza praca poświęcona jest opracowaniu oprogramowania komputerowego umożliwiającego projektowanie i wykonywanie płyt PCB metodą grawerowania ścieżek. Jest to wygodna i szybka metoda prototypowania. Wdrożenie takiego systemu prototypowania na Akademii Techniczno-Humanistycznej w Bielsku-Białej pozwoliłoby wykonywać za darmo wiele ciekawych projektów elektronicznych. Mogliby z tego skorzystać zarówno studenci jak i pracownicy ATH.

Pierwszy rozdział pracy poświęcony jest programowaniu robotów przemysłowych z naciskiem na programowanie w trybie off-line. Kolejne dwa rozdziały przedstawiają pokrótce stanowiska badawcze, na których dokonywana była walidacja opracowanego interfejsu. Rozdział czwarty stanowi instrukcję użytkowania prezentowanej aplikacji *PCB CAM Processor*, która jest głównym przedmiotem pracy. Kolejny rozdział przedstawia budowę wewnętrzną aplikacji. Ta część może być ciężka dla czytelnika, dla którego tematy związane z programowaniem obiektowym są obce. Rozdział szósty przedstawia szczegółowo obliczenia prowadzone w aplikacji oraz wykorzystywane algorytmy geometryczne. Pracę zamykają rozdziały poświęcone prezentacji wyników pracy stanowisk badawczych oraz zawierające pewne wnioski nasuwające się podczas badań. Praca została dodatkowo wzbogacona o dodatki zawierające listingi przykładowych programów obróbkowych wygenerowanych za pomocą prezentowanej aplikacji.

Cel i zakres pracy:

- Metody off-line programowania robotów przemysłowych.
- Wykonanie pakietu oprogramowania CAD/CAM dedykowanego płytom PCB.
- Opracowanie metody integracji pakietu CAD/CAM z systemem programowania robota KUKA KR6 AGILUS
- Walidacja interfejsu system CAD/CAM – robot KUKA.

1. PROGRAMOWANIE ROBOTÓW PRZEMYSŁOWYCH

Roboty przemysłowe należą do środków elastycznej automatyzacji, co oznacza, że są programowalne. Programowaniu podlegają nie tylko same ruchy manipulatora. Dzięki zastosowaniu czujników oraz komunikacji z innymi urządzeniami, roboty mogą się dostosowywać do zmieniających się warunków pracy.

Należy pamiętać, że programowanie robotów jest zwykle jednym z wielu zagadnień związanych z automatyzacją procesu. Dany zestaw wyposażenia, w którego skład mogą wchodzić roboty, uchwyty, przenośniki, podajniki, bufory międzyoperacyjne itp. określa się terminem *gniazdo produkcyjne*. Na wyższym poziomie organizacji, gniazda mogą się łączyć w elastyczne systemy obróbkowe, kierowane za pomocą scentralizowanych układów, sterujących całym procesem produkcyjnym. Zatem, programowanie manipulatorów jest często rozpatrywane jako część szerokiego zagadnienia programowania zespołu różnorodnych maszyn połączonych w zautomatyzowany system produkcyjny. W zależności od złożoności zadania oraz od zastosowań wytworzyły się różne techniki programowania [6].

1.1. Metody programowania robotów

Rozwój technologii mikroprocesorowej spowodował wyparcie starych sekwencyjnych sterowników nowoczesnymi układami komputerowymi. Współczesne programowanie robotów opiera się w pełni na technologii komputerowej i uwzględnia wszystkie aspekty spotykane w programowaniu komputerowym.

Zgrubny podział metod programowania uwzględnia programowanie bezpośrednie (on-line) oraz programowanie pośrednie (off-line). Poniższa tabela przedstawia bardziej szczegółowy podział zaproponowany w pracy [6].

Tabela 1.1. Sposoby programowania robotów[6]

pośrednie	bezpośrednie
• języki bezpośredniego programowania • języki pośredniego programowania • programowanie za pomocą grafiki	• odwzorowanie i zapamiętanie położeń robota(teach in) • ręczne prowadzenie przy wyłączonych napędach (playback, teach in) • ręczne prowadzenie ramienia pomocniczego (playbach, teach in)

Ogólnie mówiąc, układ programowania autonomicznego (off-line) to język programowania robota, który zastał rozszerzony (dzięki grafice komputerowej), w celu umożliwienia pisania programów dla robota bez jego udziału [1].

Nieco inny, bardziej szczegółowy podział zaproponowano w artykule [8]. Wymieniono tam następujące metody programowania robotów:

- „Play-back" – programowanie używane np. podczas malowania natryskowego. Następuje tu ręczne przemieszczanie narzędzia zamocowanego na robocie po określonym torze i zapamiętanie kolejnych punktów tego toru przy określonym czasie próbkowania. W trybie normalnej pracy robot odtwarza zaprogramowany tor. Rozwiązanie to jest jednak dość trudne i mało dokładne.
- „Teach-in" – polega na ręcznym ustawieniu robota w kolejnych pozycjach, zapisaniu tych pozycji, następnie odtwarzaniu ich w zadanej kolejności z uwzględnieniem interpolacji, prędkości czy przyspieszeń w poszczególnych ruchach. Rozwiązanie to jest bardzo czasochłonne i również mało dokładne. Jest jednak często stosowane w wielu prostych aplikacjach zrobotyzowanych z uwagi na prostotę i elastyczność zastosowań.
- Programowanie w językach wysokiego poziomu – jest znacznie bardziej zróżnicowane niż w przypadku obrabiarek sterowanych numerycznie. Każdy producent robotów przemysłowych posiada własny język programowania. Metoda jest skuteczna dla prostych kształtów i aplikacji, komplikuje się jednak znacząco w przypadku złożonych modeli czy długich programów.
- Interaktywne programowanie graficzne (off-line lub on-line) wykorzystuje wirtualną rzeczywistość, w której modeluje się całe stanowiska zrobotyzowane wraz z poruszającymi się robotami. Do poruszania robotem wykorzystuje się tu takie same instrukcje jak w przypadku rzeczywistej maszyny. W zależności od potrzeb stosuje się techniki programowania „Teach-in", programowanie w językach wysokiego poziomu, a nawet proste projektowanie ścieżek narzędzia CAM. To nowoczesne rozwiązanie zwiększa dokładność zaprogramowanych torów ruchu, daje możliwość przygotowania programów sterujących poza robotem (bez straty czasu pracy maszyny) oraz pozwala wykryć kolizje robotów z otoczeniem.
- CAM (off-line) – programowanie robotów przemysłowych na podstawie modeli CAD w taki sam sposób, w jaki programuje się obrabiarki sterowane numerycznie.

Ogromną zaletą jest tu parametryzacja programów oraz fakt, że zmiana modelu CAD powoduje dostosowanie ścieżek narzędzia. Programy tego typu potrafią również optymalizować ruchy ramienia robota. Z uwagi na złożoną kinematykę, narzędzie może osiągać wybrane pozycje na wiele sposobów. Program koryguje ustawienie członów, w celu uniknięcia niekorzystnych pozycji i wygładzenia ruchu robota.

1.2. Języki programowania robotów

Podczas wieloletniego rozwoju robotyki powstało wiele języków programowania robotów. Obecnie każdy producent posiada własny język, co utrudnia pracę integratorów systemów zrobotyzowanych.

Programowanie robotów polega na tworzeniu symbolicznego opisu operacji, który jest ciągiem instrukcji specjalnego wysoko-poziomowego języka programowania. Poszczególne instrukcje tego języka opisują ruch robota, działanie chwytaków, głowic technologicznych czy wejść/wyjść robota. Współczesne języki posiadają instrukcje arytmetyczne i logiczne, jak również możliwości sterowania przebiegiem programu takie jak skoki czy pętle. Nie brakuje tu również obsługi podprogramów parametrycznych czy tworzenia własnych funkcji matematycznych. Języki programowania można podzielić na trzy kategorie [1]:

1. *Wyspecjalizowane języki manipulacyjne*. Języki programowania opracowane na podstawie zupełnie nowego języka, przeznaczonego do wykorzystywania w zastosowaniach specyficznych dla robotów. Nie zawsze można je traktować jako ogólne języki programowania. Przykładem jest język VAL, który został opracowany dla robotów Unimation Inc. Jest specjalizowanym językiem programowania robotów, nieefektywnym jako język komputerowy. Brakuje tu obsługi typów łańcuchowych, znakowych czy zmiennoprzecinkowych. Nie można przesyłać argumentów dla podprogramów. Dopiero unowocześniona wersja tego języka VAL 2 posiada już te cechy. Inny przykład takiego języka specjalnego to AL opracowany na Uniwersytecie Stanforda.

2. *Biblioteka podprogramów dla istniejącego języka ogólnego przeznaczenia*. Języki rozwinięte na podstawie istniejącego, popularnego języka komputerowego, do którego dołączono biblioteki specjalizowanych podprogramów dla robotów. Programista pisze programy np. w Pascalu, często wywołując pakiety podprogramów, które zostały wcześniej opracowane dla specyficznych potrzeb robotów przemysłowych. Przykładem

może być AR-BASIC z American Cimflex oraz Robot-Basic z Intelledex. Obydwa języki stanowią specjalizowane biblioteki dla standardowego języka BASIC. Inny język to JARS, opracowany przez Laboratorium Silników Odrzutowych NASA, który powstał na bazie Pascala.

3. *Biblioteka podprogramów dla nowego języka ogólnego przeznaczenia*. Powstają przez opracowanie nowego języka komputerowego ogólnego przeznaczenia oraz dołączenie bibliotek podprogramów dla realizacji specyficznych funkcji robotów przemysłowych. Do tej grupy języków można zaliczyć ALM, opracowany przez firmę IBM. Również język KAREL wprowadzony przez firmę GMF Robotics można zaliczyć do tej kategorii, mimo iż jest podobny do języka Pascal.

1.3. Systemy programowania robotów w trybie off-line

Programowanie off-line jest obecnie bardzo popularną metodą dla tworzenia dużych systemów zrobotyzowanych. Autonomiczne fabryki samochodów wyposażone są w setki robotów przemysłowych. Trudno wyobrazić sobie sytuację, w której człowiek wchodzi na nowo powstałą halę produkcyjną i zaczyna programować kolejne roboty metodą uczenia. Również na etapie projektowania stanowisk, potrzebny jest dokładny model 3D. Pozwala to uniknąć kolizji robotów, wyznaczyć zasięg poszczególnych jednostek oraz wizualnie ocenić poprawność ich rozmieszczenia. Współczesne systemy zrobotyzowane są projektowane w taki sposób, że wiele robotów znajduje się bardzo blisko siebie. To również utrudnia zagadnienie programowania linii produkcyjnej i zwiększa ryzyko kolizji. Nie sposób wyobrazić sobie projektowania takich układów bez nowoczesnego oprogramowania bazującego na grafice komputerowej. Programy tego typu umożliwiają zaprojektowanie, zaprogramowanie i przetestowanie linii na długo przed jej budową. W efekcie, po wejściu na gotowy obiekt programistom pozostaje jedynie kalibracja wszystkich maszyn związana z pewnymi niedokładnościami wykonania stanowisk w odniesieniu do projektu. Takie postępowanie znacznie skraca czas wdrażania nowych linii produkcyjnych oraz eliminuje wiele problemów projektowych.

Na rynku dostępnych jest wiele pakietów oprogramowania tego typu. Niemal każda firma produkująca roboty przemysłowe posiada własne symulatory. Istnieją również programy umożliwiające integracje wielu robotów różnych marek. W książce [6] podano następujący podział systemów programowania off-line:

1. Ze względu na strukturę programu:
    - systemy z wirtualnym kontrolerem, które umożliwiają kontrolę programu w rzeczywistych warunkach, przy pomocy modeli 3D,
    - systemy bez wirtualnego kontrolera robota.
2. Ze względu na rodzaj obsługiwanych przez robota procesów:
    - programy specjalizowane dla robotów spawających, malujących, skrawających czy paletyzujących,
    - programy uniwersalne, umożliwiające programowanie całych szeregów działań, takich jak manipulowanie i inne czynności wytwórcze.
3. Ze względu na uniwersalność w odniesieniu do programowanych urządzeń:
    - programy dedykowane dla urządzeń konkretnego producenta,
    - uniwersalne, mogące integrować urządzenia różnych producentów.

W tabeli 1.2. zestawiono wybrane systemy off-line programowania robotów przemysłowych różnych producentów. Są to programy dedykowane robotom tej samej firmy. Systemy firm KUKA oraz ABB posiadają modułową budowę.

Przedstawione programy posiadają szereg zalet:

- możliwość projektowanie złożonych systemów zrobotyzowanych w wygodnym środowisku graficznym 3D,
- weryfikacja działania programu robota za pomocą wizualizacji działania robota w wirtualnej przestrzeni graficznej,
- weryfikacja rozmieszczenia robotów w gnieździe oraz ocena ich przestrzeni roboczej,
- możliwość wykrycia i uniknięcia kolizji robotów z innymi jednostkami lub z otoczeniem robota,
- tworzenie bardzo złożonej trajektorii narzędzia, trudnej lub nie osiągalnej metodami programowania przez nauczanie,
- osiąganie bardziej precyzyjnych ścieżek narzędzia w odniesieniu do ręcznego programowania przez nauczanie
- możliwość wykorzystywania i modyfikacji już istniejących programów,
- importowanie modeli 3D obiektów z programów graficznych,
- automatyczne generowanie kodu programu dla robota.

Tabela 1.2. Zestawienie systemów off-line do programowania robotów przemysłowych[6]

Lp.	Producent	Nazwa programu	Adres strony www
1.	Motoman	MotoSim EG	http://www.motoman.eu/uk/Products/Software/MotoSim-EG/
2.	KUKA	KUKA.Sim Pro KUKA.Sim Layout KUKA.Sim Viewer KUKA.OfficeLite	http://www.kuka-robotics.com/en/products/software/kuka_sim/start.htm
3.	ABB	ABB Robot Studio ABB Robot Aplication Builder ABB WebWare Server ABB Arc welding Die Casting Assembly Packaging	http://www.abb.com/product/pl/9AAC111580.aspx?country=00
4.	FANUC Robotics	Robogude Win OLPC	http://www.fanucrobotics.co.uk/products/pc/list.asp?cid=2&sid=2
5.	KAWASAKI	PC Roset	http://www.kawasakirobotics.com/Products/?page=otherPCROSET
6.	NACHI	AX ON DESK	http://www.nachirobotics.com.au/Content_Common/pr-Simul_AxOnDesk.seo
7.	COMAU	Smart Payload	http://www.comau.com/index.jsp?ixPageId=270
8.	IEEE	IEEE-Xplore	http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04384806
9.	CAMELOT	ROPSIM	http://www.camelot.dk/Offvson.aspx
10.	ROBOT MASTER	CASE STUDY	http://www.robotmaster.com/pdf/Software%20speeds%20programming.pdf
11.	DENSO	WINCAPS III	http://www.densorobotics.com/products_software.php
12.	AMROSE ROBOTICS	MAERSK SEALAND	http://www.amrose.dk/side.asp?ID=144

1.3.1 Roboguide

ROBOGUIDE SimPRO to produkt wprowadzony przez firmę Fanuc Robotics. Program jest kompleksowym środowiskiem graficznym służącym do tworzenia, modyfikacji oraz testowania aplikacji zrobotyzowanych. Program jest dedykowany dla konkretnego producenta robotów, posiada więc obszerną bazę robotów firmy Fanuc wraz z różnego rodzaju oprzyrządowaniem w postaci chwytaków i innych narzędzi oraz pozostałych urządzeń niezbędnych do realizacji procesu technologicznego, takich jak przenośniki, podajniki, stoły obrotowe itp. Projektant może bardzo wiernie odwzorować środowisko pracy robota, co pozwala na bardzo dokładne sprawdzenie poprawności pracy stanowiska zrobotyzowanego oraz wykrycie wszelkich nieprawidłowości, takich jak kolizje itp. Wszelkie programy tego typu umożliwiają również import elementów z zewnętrznych plików graficznych, w przypadku gdy brakuje komponentów w bazie oprogramowania. Zaletą takiego środowiska jest również możliwość optymalizacji czasowej danego stanowiska, poprzez testowanie wielu możliwych ścieżek robota i wybór takiej, która uzyskuje najlepszy czas. Wybraną ścieżkę można zaprezentować decydentom w postaci animacji. Decyzja o wprowadzeniu danego stanowiska może nastąpić po dokładnej analizie danego rozwiązania, co pozwala uniknąć straty czasu oraz kosztów nietrafionych inwestycji.

1.3.2 Win OLPC i Win OLPC+

Programy Win OLPC oraz Win OLPC+ stanowią innego rodzaju narzędzia do programowania robotów przemysłowych. Można je zaliczyć do programów bez wirtualnego kontrolera, nie przekazują one informacji w postaci geometrycznej, nie umożliwiają podglądu systemu zrobotyzowanego w rzeczywistości wirtualnej. Narzędzia te są znacznie uboższe w funkcjonalności w porównaniu do zaawansowanych systemów programowania, za to ich cena jest zdecydowanie niższa, co przy prostych projektach może mieć kluczowe znaczenie. Podczas projektowania danego systemu zrobotyzowanego może się okazać, że nie ma potrzeby stosowania drogiego i bardzo skomplikowanego systemu programowania, ponieważ zadania wykonywane przez roboty są proste i powtarzalne. W takim przypadku dużo bardziej opłacalne jest stosowanie właśnie takich narzędzi jak Win OLPC czy Win OLPC+. Programy umożliwiają tworzenie i modyfikację programów sterujących oraz ich przesyłanie między komputerem, a robotem. Jest to również program dedykowany robotom firmy Fanuc. Wersja rozszerzona umożliwia dodatkowo podgląd punktów charakterystycznych w przestrzeni 3D, co ułatwia pracę programiście.

1.3.3. PC-ROSET

Jest to kolejny system stworzony do programowania robotów, dedykowany danej marce robotów, którą jest w tym przypadku Kawasaki. Oprogramowanie stanowi symulator rzeczywistego kontrolera robota. Na ekranie widoczny jest panel operatorski, za którego pomocą można sterować robotem i uczyć się jego obsługi. Za pomocą oprogramowania można [6]:

- symulować trajektorie ruchów robota,
- opracowywać programy w trybie off-line,
- badać kolizje i optymalizować ścieżki ruchu,
- weryfikować przestrzeń roboczą,
- sprawdzać czasy trwania cykli pracy robota,
- korzystać z bogatej biblioteki robotów i oprzyrządowania,
- przesyłać programy do robota oraz zgrywać je na PC,
- importować elementy 3D z programów graficznych,
- kontrolować porty wejść/wyjść kontrolera robota.

Program dostępny jest w trzech wersjach:

- handling – wersja ogólnego przeznaczenia z obszerną biblioteką komponentów
- paint – wersja dedykowana aplikacjom malarskim
- arc – dedykowana pracom spawalniczym

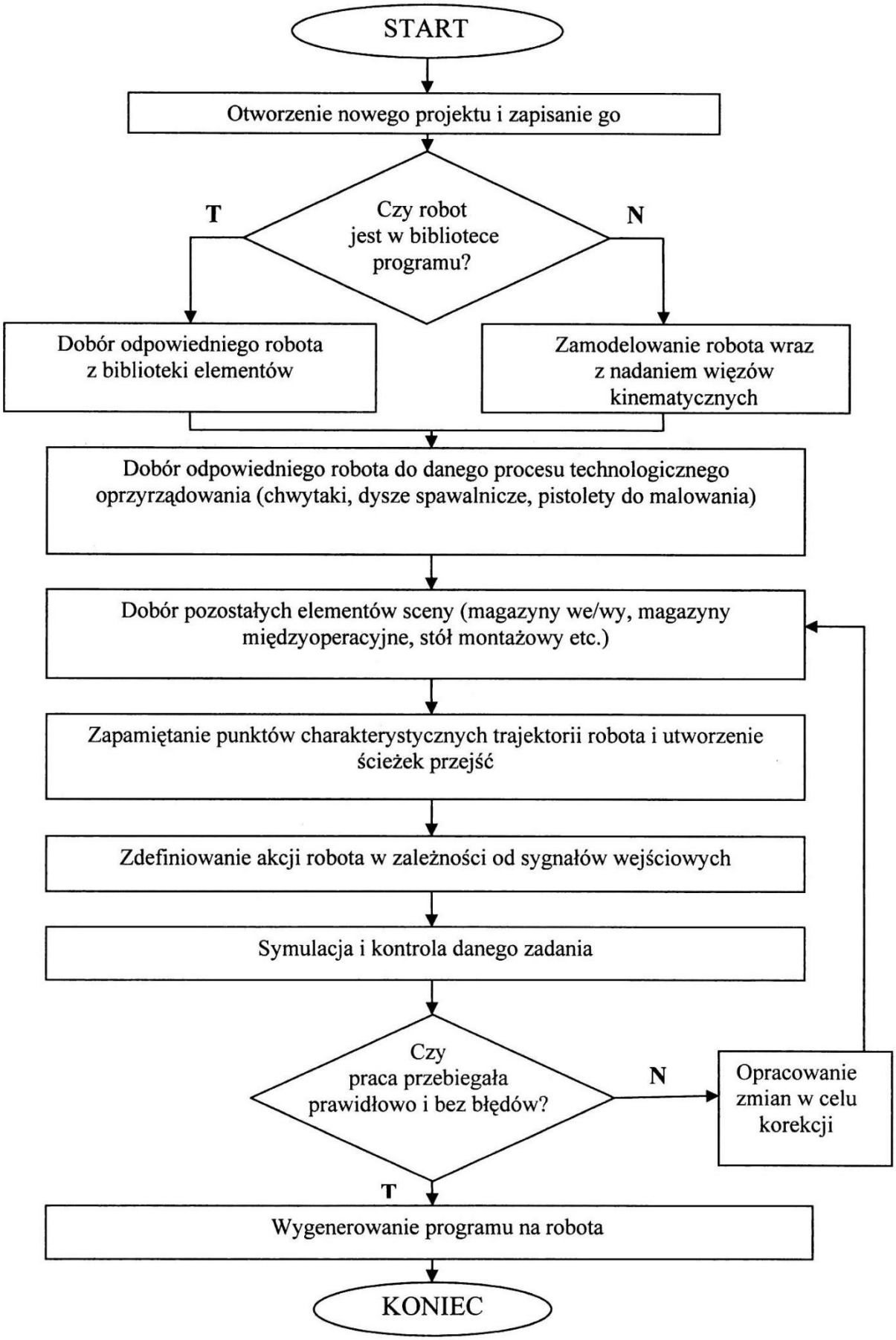
1.3.4. KUKA RobotSim

Firma Kuka, tak jak inni czołowi producenci na rynku robotów przemysłowych, wyszła na przeciw rosnącym wymaganiom rynkowym. Każda minuta zaoszczędzona w procesie wdrażania nowych stanowisk zrobotyzowanych lub przezbrajaniu już istniejących zamienia się na wyraźne korzyści finansowe. KUKA RobotSim jest pakietem bardzo zaawansowanego systemu wspomagania planowania, projektowania i programowania zrobotyzowanych gniazd wytwórczych. System nie ma zbyt wygórowanych wymagań sprzętowych, może więc pracować na zwykłym współczesnym komputerze PC. Cena tego pakietu, jak i pozostałych dostępnych na rynku, jest jednak bardzo wysoka. Firma proponuje narzędzia zróżnicowane co do ceny i możliwości, zależnie od potrzeb klientów:

- Pakiet KUKA OfficeLite jest odzwierciedleniem panelu sterowania robota. Umożliwia otwieranie wcześniej opracowanych programów, edycję, symulację wejść i wyjść oraz monitorowanie sygnałów zewnętrznych. KUKA OfficeLite przeznaczone jest przede wszystkim do pisania i weryfikacji programów bez konieczności korzystania z rzeczywistego kontrolera robota.
- Pakiet Sim Layout umożliwia edycję wcześniej napisanych programów. Jego zaletą jest wizualizacja środowiska pracy robota.
- Kuka Sim Viewer jest jedynie przeglądarką zaprojektowanych już wcześniej stanowisk zrobotyzowanych. Umożliwia ich edycję w zakresie zmiany takich parametrów jak, prędkość czy kąty obrotu osi robota.
- WinRobSim jest bardzo podobny do Sim Viewer. Oprócz wizualizacji i zmiany podstawowych parametrów umożliwia zarządzanie zachowaniem robota w odniesieniu do sygnałów zewnętrznych oraz rejestracje gotowych symulacji.
- KUKA SimPro jest najbardziej rozbudowanym, profesjonalnym pakietem, zawierającym wszystkie powyższe moduły oraz ich funkcjonalności. Zawiera bogatą bibliotekę komponentów i robotów oraz daje możliwość importu dowolnych brył z programów graficznych. Umożliwia kompleksowe projektowanie, programowanie i testowanie stanowisk. Ponadto daje możliwość integracji robotów z zewnętrznymi urządzeniami za pomocą portów wejścia/wyjścia oraz sieci przemysłowych. Umożliwia ocenę poprawności rozmieszczenia robotów, wykrywanie kolizji i innych niepożądanych sytuacji. Daje również możliwość zapisania symulacji jako animacji osadzonej np. w pliku formatu PDF [13,14].

1.3.5. Algorytm postępowania

Niezależnie od wyboru oprogramowania do pracy z robotami w trybie off-line, należy mieć na uwadze pewien ogólny zarys postępowania podczas projektowania stanowisk zrobotyzowanych. Ogólna znajomość tego zagadnienia i całościowe podejście do tematu ułatwia zrozumienie istoty działania wybranego oprogramowania oraz umożliwia efektywniejszą pracę programisty. Kolejność działań w procesie projektowania stanowiska zrobotyzowanego została zaproponowana w książce [6] w formie algorytmu pokazanego poniżej.



Rysunek 1.1. Algorytm postępowania przy tworzeniu programu w trybie off-line[6]

1.4. Główne problemy związane z systemami programowania off-line

Zagadnienie budowy systemów programowania autonomicznego wiąże się z rozwiązaniem wielu problemów. Twórcy tego typu oprogramowania powinni zwracać szczególną uwagę na takie aspekty jak [5]:

- *Sposób komunikacji użytkownika z układem.* Głównym celem podczas rozwoju tego typu programów jest stworzenie narzędzia ułatwiającego programowanie robotów. Jednocześnie należy zapewnić możliwość programowania bez udziału realnej maszyny. Te dwa założenia są nieco sprzeczne, ponieważ programowanie manipulatorów jest trudne, a bez udziału robota ma stać się łatwiejsze. Kluczowym problemem w programowaniu jest znajomość języka programowania danego robota. Jest to dużym utrudnieniem dla niewykwalifikowanego personelu. Systemy programowania autonomicznego powinny się komunikować z użytkownikiem na innym, prostszym poziomie, z wykorzystaniem grafiki 3D. Głównym celem pracy jest nauczenie wirtualnej maszyny wykonywania ruchu w przestrzeni z wykorzystaniem sześciu stopni swobody. Interfejs użytkownika powinien umożliwiać łatwe przemieszczenie manipulatora za pomocą myszy lub innych urządzeń wskazujących, co daje możliwość tworzenia większości aplikacji zrobotyzowanych osobom nie będącym programistami. Bez znajomości języka programowania można osiągnąć ten sam efekt i do tego w krótszym czasie. Tłumaczeniem sekwencji ruchów na język programowania zajmuje się tutaj sam system programowania autonomicznego. Użytkownik nie musi rozumieć kodu programu, wystarczy że będzie potrafił załadować program do robota i uruchomić go na maszynie. Potrzebna jest zatem tylko wiedza z zakresu obsługi danego robota przemysłowego.
- *Trójwymiarowa grafika.* Nieodzownym elementem systemu programowania autonomicznego jest zastosowanie graficznej reprezentacji symulowanego robota oraz całego jego otoczenia. Wszystkie obiekty, takie jak: robot, uchwyty, obiekty manipulacji, przenośniki czy podajniki powinny być zamodelowane w trójwymiarowej grafice komputerowej. Aby przyspieszyć pracę z takimi systemami, pożądane jest wykorzystywanie modeli zaprojektowanych w systemach modelowania CAD. Musi zatem istnieć możliwość importu plików w wielu popularnych formatach graficznych. Istotne jest również to, aby możliwe było edytowanie tych elementów już wewnątrz systemu lub nawet tworzenie prostych brył dla przyspieszenia tworzenia stanowiska zrobotyzowanego. Zintegrowanie systemów CAD, CAM oraz narzędzi programowania w jeden system daje jeszcze większe możliwości. Zmiana modelu CAD, może mieć wpływ na model stanowiska oraz na

programy obróbkowe. Większość zmian powinna się dokonywać automatycznie.

Ważnym aspektem, znanym z grafiki komputerowej, stosowanym w grach, jest wykrywanie kolizji. Jest to potrzebne zarówno do ostrzegania przed sytuacjami mogącymi doprowadzić do uszkodzenia robota, jak również do umożliwiania prawidłowej manipulacji obiektów. Zamodelowany robot będzie używał różnego rodzaju chwytaków do pobierania detali, będzie również odkładał je na przenośniki. To również wymaga mechanizmów wykrywania kolizji obiektów, która w tym przypadku jest pożądana. Z uwagi na pewne niedokładności modeli, system musi prowadzić obliczenia z pewną tolerancją, aby zamodelowanie uchwycenia przedmiotu było możliwe.

- *Emulacja kinematyczna – planowanie trajektorii.* Ważnym aspektem jest poprawne uwzględnienie kinematyki manipulatora. Algorytm wyznaczania trajektorii robota zaimplementowany w systemie programowania, powinien być zgodny z tym stosowanym w sterowniku robota. Model stanowiska bardzo precyzyjnie określa wymiary i położenie poszczególnych elementów. Może więc dojść do sytuacji, że na modelu robot nie ma kolizji z otoczeniem, natomiast rzeczywista aplikacja, w której sterownik stosuje inny algorytm wyznaczania ścieżki między dwoma punktami, spowoduje kolizję. Jest to oczywiście niedopuszczalne. Programiści robotów wiedzą, że ruchy typu point-to-point nie mają ustalonej ścieżki i należy ich unikać, szczególnie na duże odległości oraz blisko innych elementów stanowiska. Niedoświadczony użytkownik, który korzysta z systemu programowania może nie mieć takiej świadomości, dlatego system powinien gwarantować wierność symulacji z rzeczywistością. Oprogramowanie powinno również umożliwiać optymalizację ścieżki robota oraz unikanie pozycji osobliwych.
- *Emulacja dynamiczna.* Symulacja ruchu robota może pomijać właściwości dynamiczne, jeżeli system programowania poprawnie wyznacza trajektorię, a rzeczywisty robot porusza się po zadanym torze z pomijalnymi błędami. Jednakże przy dużych szybkościach i obciążeniach, błędy realizacji trajektorii mogą mieć istotne znaczenie. Konieczne staje się modelowanie dynamiki manipulatora oraz obiektów przez niego przenoszonych, jak również algorytmu sterowania wykorzystywanego w sterowniku robota.
- *Symulacja wieloprocesowa.* Większość zrobotyzowanych stanowisk zawiera więcej niż jednego robota. Posiada również inne urządzenia, takie jak przenośniki czy obrabiarki. Z tego powodu ważne jest, aby system programowania był zdolny do symulowania wielu poruszających się urządzeń oraz prowadzenia innych działań równoległych. Konieczne jest tutaj umożliwienie pisania odrębnych programów dla każdego urządzenia oraz wyko-

nywanie tych programów równolegle. Dodanie instrukcji oczekiwania oraz przesyłania sygnałów umożliwia symulacje współpracy wszystkich urządzeń, które wchodzą w skład stanowiska.

- *Symulacja czujników.* Duża część programu sterującego robota obejmuje sprawdzanie błędów, obsługę wejść/wyjść i inne czynności wymagające interakcji z otoczeniem. Stąd ważna jest zdolność systemu programowania do wiernego modelowania otoczenia, która umożliwia symulację pełnego zastosowania, obejmującego współpracę z urządzeniami wejścia/wyjścia, czujnikami oraz komunikację z innymi urządzeniami.
- *Tłumaczenie językowego do systemu docelowego.* Dla obecnych użytkowników robotów przemysłowych, kłopotliwe jest to, że każdy producent robotów dostarcza własny język programowania. Integratorzy, którzy uruchamiają nowe stanowiska, a chcą korzystać z różnych marek robotów, muszą znać wiele języków. Oczywistym ułatwieniem programowania są uniwersalne systemy autonomicznego programowania, które posiadają postprocesory dla popularnych języków programowania robotów. Takie rozwiązanie umożliwia szybkie tworzenia oprogramowania sterującego dla robotów różnych producentów, bez znajomości stosowanych tam języków. Ponadto istnieje wówczas możliwość integracji różnych robotów na jednym stanowisku, ponieważ wewnątrz systemu programowania wszystkie roboty programuje się w taki sam sposób.
- *Wzorcowanie gniazd produkcyjnych.* Komputerowe modelowanie rzeczywistych sytuacji wiąże się zawsze z pewnymi uproszczeniami oraz niedokładnością odwzorowania rzeczywistości. Aby zastosować programy wygenerowane w systemie programowania off-line, należy dopasować do siebie model rzeczywisty z tym utworzonym w systemie. Odbywa się to najczęściej poprzez definiowanie baz, czyli układów współrzędnych związanych z przedmiotem obrabianym lub stanowiskiem pracy. Odnoszenie całego programu sterującego do jednego układu współrzędnych jest efektywne, ponieważ wdrożenie robota wymaga jedynie nauczenia nowej bazy i ustalenia układu związanego z narzędziem. Program sterujący pozostaje niezmieniony i może być zastosowany na wielu podobnych stanowiskach.

2. STANOWISKO BADAWCZE – ROBOT KUKA

2.1. Opis stanowiska

Stanowisko do prowadzenia obróbki PCB metodą grawerowania składa się z robota przemysłowego *KUKA Agilus KR6 R900 sixx* oraz stolika laboratoryjnego. Na flandze robota został zainstalowany specjalny uchwyt, umożliwiający zamocowanie szlifierki prostej *DWT GS06-27* pełniącej w tym przypadku funkcję elektrowrzeciona. Szlifierka została wyposażona w tulejkę zaciskową o średnicy 3.175mm (1/8 cala). Jest to średnica dedykowana narzędziom stosowanym przy wykonywaniu PCB. Wiertła oraz frezy grawerskie nienależnie od średnicy roboczej, posiadają taką właśnie średnicę trzonka. Poniżej przedstawiono przykładowe narzędzia grawerskie oraz zarys ostrza narzędzia stosowanego w testach.

Rysunek 2.1. Narzędzia grawerskie o różnym kącie wierzchołkowym oraz stosowane w testach ostrze

Jak pokazano powyżej istnieje wiele dostępnych na rynku rodzajów narzędzi grawerskich, zróżnicowanych co do kąta wierzchołkowego. Używane w testach narzędzie posiadało kąt wierzchołkowy 15° oraz szerokości czoła 0.1mm.

Rysunek 2.2. Szlifierka prosta *DWT GS06-27* – źródło: producent

2.2. Przygotowanie stanowiska

Po przytwierdzeniu laminatu PCB do stolika laboratoryjnego, należy wyznaczyć układ współrzędnych, leżący na powierzchni laminatu. Z zamocowanym narzędziem grawerskim należy zbliżyć się do powierzchni laminatu i metodą z użyciem papieru, sprawdzać odległość. Po dostatecznym zbliżeniu się do powierzchni laminatu, w punkcie stanowiącym początek nowego układu współrzędnych, należy zapisać pozycję robota. Następnie powtórzyć to dla kolejnych dwóch punktów. W pierwszej kolejności należy osiągnąć punkt leżący na osi X nowego układu współrzędnych, następnie wskazać punkt na płaszczyźnie XY z dodatnią wartością Y. Wszystkie punkty muszą oczywiście leżeć na płaszczyźnie roboczej, gdyż są one potrzebne do geometrycznego jej wyznaczenia. Przedstawiona metoda wyznaczania nowej bazy nosi nazwę *trójpunktowej* [11].

Pomiar bazy, czyli nowego układu współrzędnych daje użytkownikowi następujące korzyści [10]:

- Przesuw wzdłuż krawędzi elementu obrabianego:

  TCP może być ręcznie przesuwany wzdłuż krawędzi powierzchni roboczej lub obrabianego elementu.

- Układ współrzędnych odniesienia:

  Wczytane punkty odnoszą się do wybranego układu współrzędnych.

- Korekta/przesunięcie układu współrzędnych:

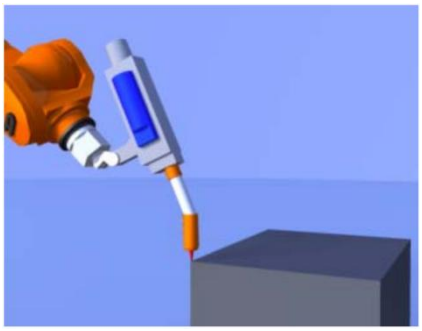
  Możliwe jest namierzanie punktów w odniesieniu do podstawy. Jeśli podstawa musi zostać przesunięta, np. ze względu na przesunięcie powierzchni roboczej, punkty ulegają automatycznemu przesunięciu i nie ma potrzeby ich ponownego namierzania.

- Użycie kilku układów współrzędnych podstawy:

  Można utworzyć do 32 różnych układów współrzędnych i używać ich zależnie od etapu przebiegu programu.

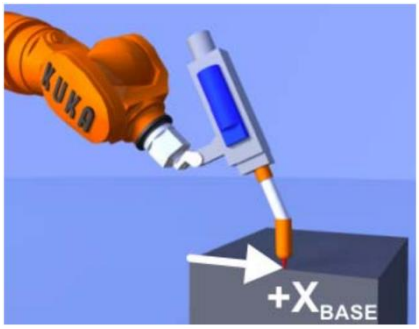
Istnieją trzy możliwości wyznaczenia bazy robota KUKA. Należą do nich: metoda trójpunktowa, metoda pośrednia oraz ręczne wprowadzenie przesunięcia względem uniwersalnego układu współrzędnych. Poniżej przedstawiono szczegółowo przebieg procedury wyznaczenia bazy metodą trójpunktową [10]:

1. Wybrać w menu głównym **Uruchomienie > Pomiar > Baza > 3-punkty**.
2. Podać numer i nazwę bazy. Potwierdzić za pomocą **Dalej**.
3. Wprowadzić numer narzędzia, którego TCP jest używany do pomiaru bazy. Potwierdzić za pomocą **Dalej**.
4. Punktem odniesienia narzędzia (TCP) najechać na punkt początkowy nowej podstawy. Nacisnąć przycisk **Wymierz** i potwierdzić pozycję, naciskając **Tak.**



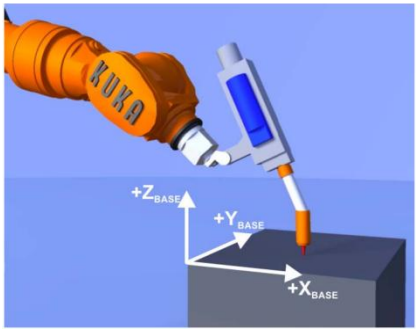
Rysunek 2.3. Pierwszy punkt: Punkt początkowy[10]

5. Przesunąć punkt odniesienia narzędzia (TCP) do dowolnego punktu na dodatniej osi X nowej podstawy. Nacisnąć **Wymierz** i potwierdzić pozycję, naciskając **Tak.**



Rysunek 2.4. Drugi punkt: kierunek X[10]

6. Przesunąć punkt odniesienia narzędzia (TCP) do dowolnego punktu na płaszczyźnie XY z dodatnią wartością Y. Nacisnąć **Wymierz** i potwierdzić pozycję, naciskając **Tak.**



Rysunek 2.5. Trzeci punkt: płaszczyzna XY[10]

7. Nacisnąć **Zapisz**.
8. Zamknąć menu.

Przedstawiona powyżej metoda jest prosta, szybka i wygodna. Mając odrobinę wprawy w obsłudze robota za pomocą KUKA SmartPad, czyli ręcznego programatora wyposażonego w sześcioosiową mysz, można naprawdę efektywnie poruszać ramieniem. Stanowi to olbrzymią przewagę nad robotami starszej generacji lub współczesnymi robotami nieposiadającymi takich rozwiązań. Roboty firmy Kawasaki oraz wielu innych producentów obsługuje się za pomocą zwykłych przycisków służących do przejazdów względem wybranej osi układu współrzędnych. Wymaga to ciągłego skupienia, osoba obsługująca maszynę musi pamiętać, w którym kierunku zorientowany jest układ współrzędnych. Stanowi to szczególne utrudnienie, gdy prace prowadzone są z różnych stron ramienia. KUKA wyszła naprzeciw również i temu problemowi. Użytkownik może w graficzny sposób wybrać, gdzie się aktualnie znajduje względem robota. Jeżeli wybierze poprawnie, ruchy ramienia oraz zmiany orientacji narzędzia będą odzwierciedlać ruchy i wychylenia sześcioosiowej myszy umiejscowionej na prawej powierzchni bocznej programatora ręcznego.

2.3. KRL – język programowania robota

KRL jest językiem wysokiego poziomu podobnym do języków PASCAL czy BASIC. Posiada wiele typów geometrycznych, zapisywanych podobnie jak struktury w językach C/C++ czy Java, służących do zapisu punktów oraz układów współrzędnych. Obsługuje parametryczne programy oraz funkcje. Posiada rozbudowaną składnię nagłówków plików oraz zapisu poszczególnych instrukcji ruchu.

Dla uproszczenia programu głównego stosowanego na stanowisku, zostały wprowadzone podprogramy umożliwiające wykonanie szybkiego lub roboczego przejazdu w wybranej interpolacji oraz podprogram wykonujący wiercenie. Wszystkie parametry dotyczące ruchu tj.: prędkość, przyspieszenie itp. zostały określone wewnątrz tych podprogramów. Takie podejście czyni program główny znacznie czytelniejszym i krótszym. Szczegóły dot. wspomnianych podprogramów oraz przykładowe programy wiercenie oraz grawerowania zostały przedstawione w dodatkach A, B oraz C niniejszej pracy.

Więcej szczegółów nt. programowania robotów KUKA w języku KRL znajduje się w instrukcjach producenta w języku polskim [10] i [11] oraz w angielskojęzycznej, pełnej dokumentacji KRL, umożliwiającej efektywne programowanie na poziomie eksperta [9].

3. STANOWISKO BADAWCZE – TOKARKA Z UKŁADEM STEROWANIA SINUMERIK

3.1. Opis Stanowiska

Stanowisko badawcze składa się z tokarki wyposażonej w układ sterowania Sinumerik oraz elektrowrzeciona o maksymalnej prędkości obrotowej 12000 obr/min zamontowanego zamiast imaka narzędziowego. Poniżej przedstawiono ogólny wygląd stanowiska.

Rysunek 3.1. Stanowisko - badawcze tokarka z układem sterowania Sinumerik

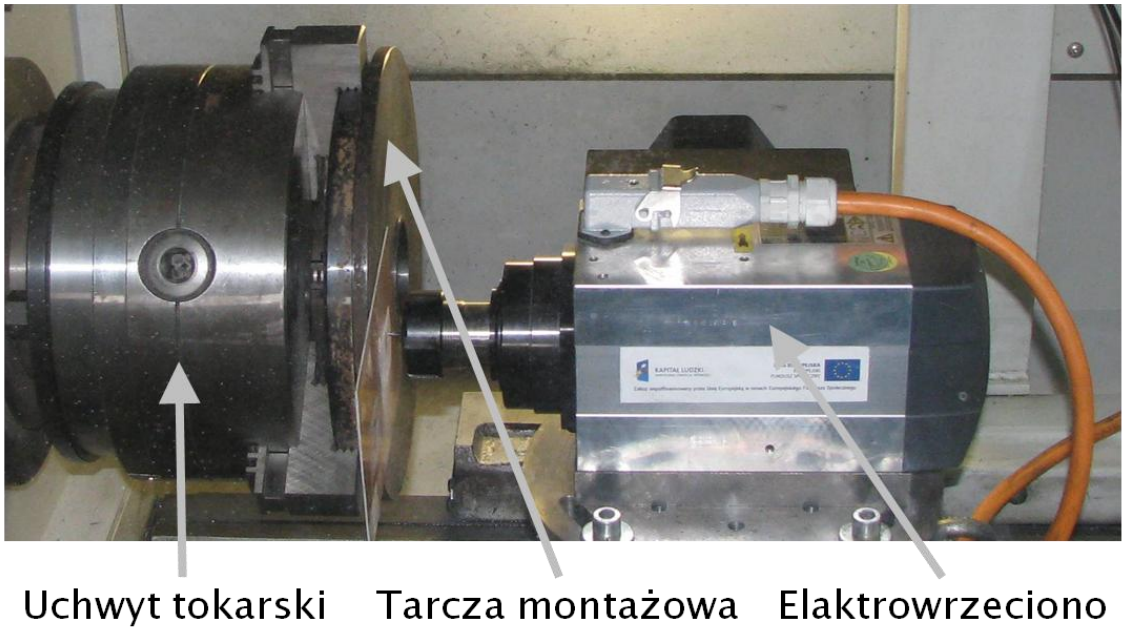
Poniżej przedstawiono sposób zamocowania elektrowrzeciona.

Rysunek 3.2. Elektrowrzeciono zamocowane na saniach poprzecznych

3.2. Sposób mocowania laminatu PCB

Laminat PCB mocowany jest na stalowej tarczy zamocowanej w uchwycie trójszczękowym obrabiarki. Istotne jest aby tarcza posiadała płaską powierzchnię czołową oraz po zamocowaniu leżała w płaszczyźnie XY maszyny. Kalibracji można dokonać za pomocą czujnika zegarowego.

Z uwagi na bardzo małe siły skrawania, laminat mocowany jest do tarczy za pomocą silnej taśmy dwustronnej. Oczywiście należałoby wykonać odpowiedni uchwyt, umożliwiający takie mocowanie, które daje możliwość wiercenia na wylot laminatu, a nawet odwrócenia dla wykonywania płytek dwustronnych. Nie należało to jednak do celów pracy. Badania na tokarce zostały rozpoczęte dużo później, jako dodatkowy punkt pracy, z uwagi na niezadawalające wyki pracy robota.



Rysunek 3.3. Mocowanie laminatu PCB na tarczy montażowej

3.3. Funkcja TRANSMIT – obróbka frezarska na częściach toczonych

Programowanie:

```
TRANSMIT albo TRANSMIT(n)
TRAFOOF
```

Tabela 3.1. Objaśnienie poleceń [15]

`TRANSMIT`	Uaktywnia pierwszą uzgodnioną funkcję TRANSMIT
`TRANSMIT(n)`	Uaktywnia n-tą uzgodnioną funkcję TRANSMIT; n może wynosić maksymalnie 2 (TRANSMIT(1) odpowiada TRANSMIT).
`TRAFOOF`	Wyłącza aktywną transformację

Funkcja TRANSMIT daje następujące możliwości [15]:

- Obróbka na stronie czołowej części toczonych w zamocowaniu dla toczenia (otwory, kontury).
- Do programowania tej obróbki można używać kartezjańskiego układu współrzędnych.
- Sterowanie transformuje zaprogramowane ruchy postępowe w kartezjańskim układzie współrzędnych na ruchy postępowe realnych osi maszyny (przypadek standardowy):
    - oś obrotowa
    - oś dosuwu prostopadła do osi obrotu
    - oś podłużna równoległa do osi obrotu
    - osie liniowe są prostopadłe do siebie
- Przesunięcie środka narzędzia w stosunku do osi obrotu jest dopuszczalne.
- Prowadzenie prędkości uwzględnia ograniczenia zdefiniowane dla ruchów obrotowych.

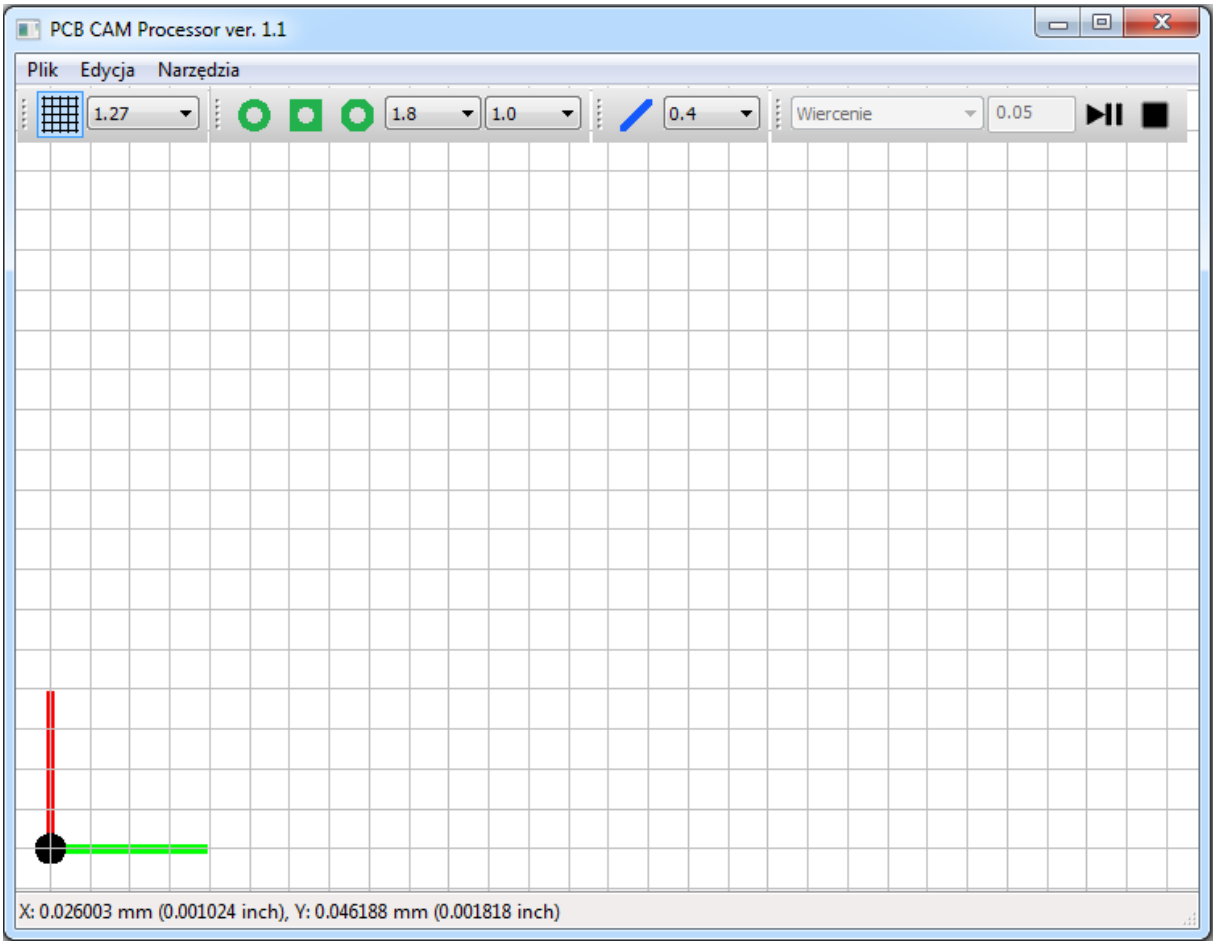
Przykładowy program z użyciem TRANSMIT [15]:

```
N10 T1 D1 G54 G17 G90 F5000 G94  Wybór narzędzia
N20 G0 X20 Z10 SPOS=45           Dosunięcie do pozycji wyjściowej
N30 TRANSMIT                     Uaktywnienie funkcji TRANSMIT
N40 ROT RPL=-45                  Nastawienie frame
N50 ATRANS X-2 Y10
N60 G1 X10 Y-10 G41              Obróbka zgrubna czopu kwadratowego
N70 X-10
N80 Y10
N90 X10
N100 Y-10
N110
...
```

4. OPIS OPROGRAMOWANIA

4.1. Moduł CAD - funkcje i możliwości

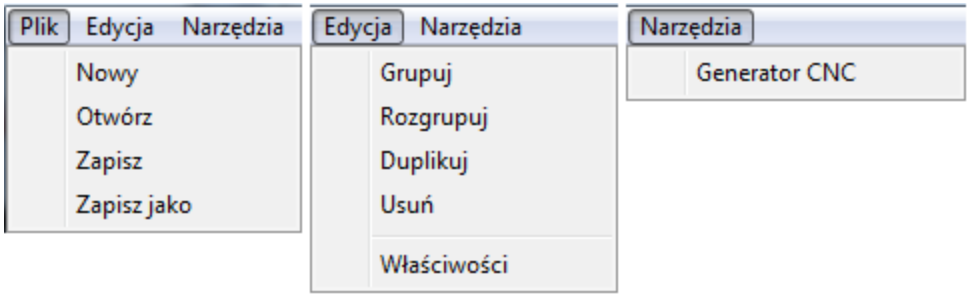
Oprogramowanie *PCB CAM Processor* umożliwia szybkie i wygodne projektowanie płyt PCB. Interfejs graficzny aplikacji pozwala na efektywne tworzenie modelu przy użyciu myszy. Podobnie jak w innych systemach CAD, kadrowanie widoku odbywa się poprzez użycie kółka myszy. Kursor wskazuje wówczas kierunek przybliżania/oddalania. Rozmieszczanie poszczególnych elementów modelu ułatwia również siatka, którą można dowolnie konfigurować lub wyłączyć jej widoczność dla zwiększenia czytelności modelu.



Rysunek 4.1. Okno główne aplikacji

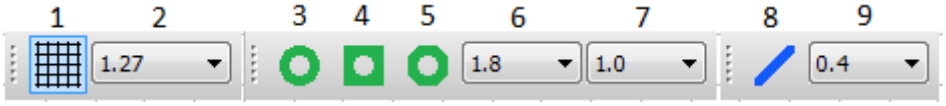
Na powyższym rysunku przedstawiono widok głównego okna aplikacji, na które składają się następujące elementy:

- Belka górna - zawiera podstawowe opcje manipulacji na plikach, opcje edycji oraz narzędzia modułu CAM



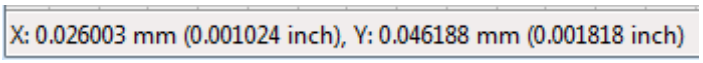
Rysunek 4.2. Menu belki górnej

- Belka rysunkowa - przybornik zawierający funkcje potrzebne do tworzenia modelu



Rysunek 4.3. Belka rysunkowa

- Pole edycyjne - interaktywny obszar służący do tworzenia modelu
  W lewym dolnym rogu widać początek układu współrzędnych z zaznaczonymi osiami X - w prawo, Y - w górę.
- Belka statusowa wyświetlająca aktualne położenie kursora myszy

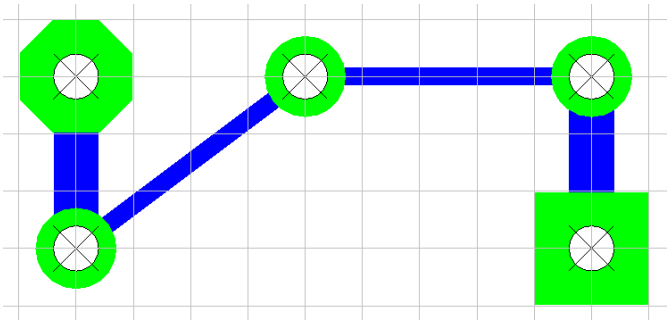


Rysunek 4.4. Fragment belki statusowej z widocznymi współrzędnymi kursora

4.1.1. Tworzenie prostego modelu

Jak wspomniano powyżej do modelowania wykorzystuje się narzędzia belki rysunkowej. Przycisk 1 służy do włączenia/wyłączenia widoczności siatki. Kolejny element na belce definiuje gęstość siatki wyrażoną w milimetrach. Przyciski 3,4 i 5 umożliwiają osadzenie na ekranie pól lutowniczych kolejno w kształcie: koła, kwadratu i ośmiokąta. Elementy 6 i 7 odpowiadają za wybór wymiaru zewnętrznego(6) oraz średnicy otworu(7) pola lutowniczego. Za pomocą przycisku 8 dodaje się do modelu ścieżki łączące pola lutownicze. Szerokość rysowanej ścieżki definiuje element 9. Aby narysować najprostszy model przy użyciu *PCB CAM Processor* wystarczy wybrać pojedynczym kliknięciem narzędzie 3,4 lub 5 następnie najechać kursorem myszy nad pole edycyjne. Każde kliknięcie w obszarze tego pola spowoduje dodanie kolejnego pola lutowniczego do aktualnie tworzonego modelu.

Podczas poruszania myszą z wybraną funkcją rysowania pod kursorem myszy widoczny jest podgląd wstawianego elementu z uwzględnieniem wybranych wcześniej wymiarów. Po dodaniu kilku pól lutowniczych o żądanych kształtach i wymiarach można połączyć pola za pomocą narzędzia 8. Narysowanie odcinka ścieżki jest równie łatwe jak dodanie pola lutowniczego. Po wybraniu z belki rysunkowej funkcji 8, należy pojedynczym kliknięciem wybrać punkt początkowy ścieżki. Następnie z widocznym podglądem wskazać punkt końcowy. Zaczepienie ścieżki w punktach zajmowanych przez pola lutownicze spowoduje trwałe powiązanie pomiędzy ścieżką, a polami. Tak skonstruowany model jest interaktywny. Oznacza to, że poruszenie jednego elementu powoduje przemieszczenie wszystkich innych z nim powiązanych. Efekt osiągnięty w kilka sekund przy użyciu opisanych narzędzi przedstawia poniższy rysunek.



Rysunek 4.5. Przykład najprostszego modelu

4.1.2. Zaznaczanie i odznaczanie wybranych obiektów

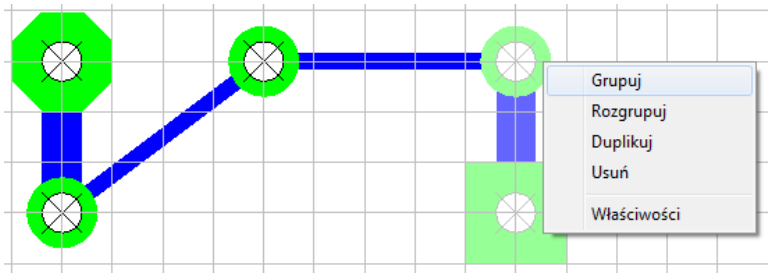
Prezentowane oprogramowanie umożliwia dwojaki sposób zaznaczania obiektów: wskazanie lewym przyciskiem myszy oraz zaznaczanie poprzez obszar prostokątny. Wskazywanie obiektów pojedynczym kliknięciem powoduje zawsze zaznaczenie jednego, ostatnio kliknięttego elementu. Używając klawisza *Shift* na klawiaturze można zaznaczyć wiele obiektów. Podczas zaznaczania kolejnych elementów klawisz ten musi być stale wciśnięty. Ponowne wskazanie obiektu za pomocą myszy z wciśniętym *Shift* spowoduje jego odznaczenie. Zaznaczanie poprzez obszar odbywa się w sposób następujący: należy wskazać lewym przyciskiem myszy na pusty obszar pola edycyjnego, celem wyboru pierwszego narożnika prostokąta, następnie trzymając wciśnięty klawisz myszy wskazać kursorem przeciwległy narożnik i puścić przycisk myszy. Podczas poruszania kursorem na ekranie widoczny jest podgląd zaznaczanego obszaru w postaci czarnej ramki. Kombinacja klawisza *Shift* i zaznaczenia prostokątnego powoduje dodania kolejnego obszaru do puli zaznaczenia, na zasadzie sumy zbiorów.

4.1.3. Przesuwanie obiektów

Aplikacja umożliwia również zmianę położenia wybranych elementów. Przesuwanie odbywa się na zasadzie prostego przeciągnięcia wybranego obiektu bądź grupy wielu zaznaczonych uprzednio obiektów. Wystarczy kliknąć i przytrzymać lewy przycisk myszy na wybranym elemencie, następnie przeciągnąć go na miejsce docelowe i puścić przycisk myszy. Podczas przemieszczanie obiektu widoczny jest podgląd ułatwiający prowadzenie tej operacji. Dzięki powiązaniom pomiędzy obiektami możliwa jest pewna interakcja. Przykładowo przesuwanie pola lutowniczego powoduje również zmianę położenia końca ścieżki związanego z tym polem.

4.1.4. Grupowanie i rozgrupowanie elementów modelu

Posiadając już fragment modelu, który jest w pewnym sensie spójny i gotowy, zachodzi potrzeba ochrony naszej pracy przed nieuważnym zniszczeniem. Prezentowana aplikacja dostarcza opcję grupowania obiektów. Fragment modelu, który został zgrupowany jest teraz traktowany jako jeden element. Próba przesunięcia jednego członka powoduje przemieszczenie całej grupy. Aby skorzystać z funkcji grupowania należy wywołać menu kontekstowe prawego przycisku myszy, klikając na jednym z uprzednio zaznaczonych obiektów. Można również użyć menu *Edycja* z belki górnej (Rysunek 4.2.).



Rysunek 4.6. Przykład grupowania obiektów

Powyższy rysunek prezentuje użycie funkcji grupowania wywołanej z menu kontekstowego prawego przycisku myszy. Obiekty poddane grupowaniu zostały uprzednio zaznaczone co jest widoczne w aplikacji jako rozjaśnienie.

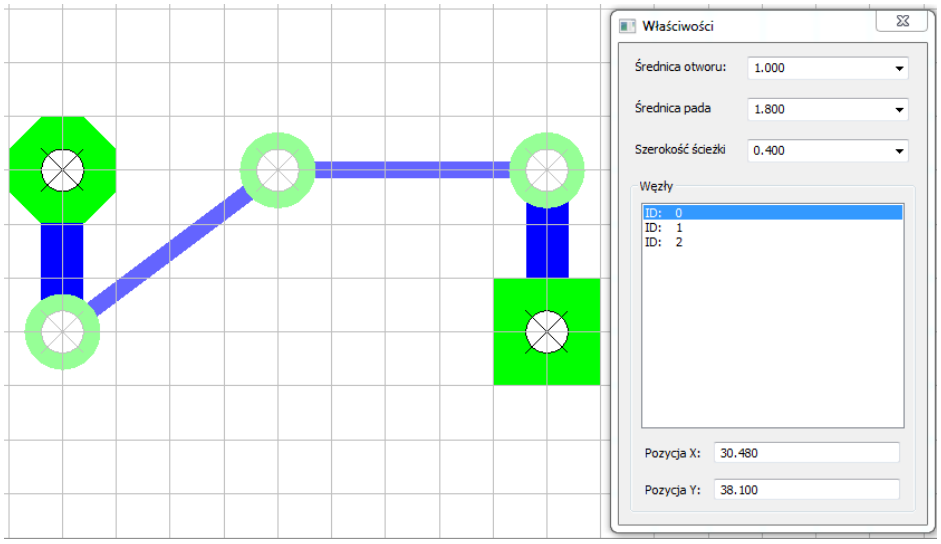
Rozgrupowanie odbywa się w analogiczny sposób i powoduje powrót modelu do poprzedniej postaci. Grupowanie może odbywać się etapowo. Grupy mogą wchodzić w skład kolejnych większych grup. Powstaje w ten sposób pewne drzewko zależności pomiędzy grupami. Nie ma ograniczeń ilości poziomów takiego układu. Budowa i zasada działania tego mechanizmu została szczegółowo omówiona w dalszej części pracy.

4.1.5. Duplikowanie i usuwanie obiektów

Kolejną operacją edycji jest duplikowanie. Podobnie jak w przypadku grupowania opcja dostępna jest w menu kontekstowym prawego przycisku myszy (Rysunek 4.6.) lub w menu *Edycja* na belce górnej (Rysunek 4.2.). Po wybraniu tej opcji pod kursorem myszy pojawi się kopia zaznaczonych obiektów. Aby wstawić obiekty należy wskazać wybrane miejsce i nacisnąć lewy przycisk myszy. Anulowanie operacji jest możliwe poprzez naciśnięcie prawego przycisku myszy lub klawisza *Esc* na klawiaturze. Duplikacja jest szybkim i wygodnym sposobem na tworzenie dużych modeli, w których powtarzają się pewne fragmenty. W połączeniu z funkcją grupowania znacznie przyspiesza projektowanie. Obecnie w aplikacji brak bazy obudów elementów elektronicznych, dlatego gdy raz zamodeluje się np. obudowę układu scalonego warto zgrupować wszystkie pola lutownicze należące do niej, by później jednym kliknięciem dokonywać jej duplikacji i swobodnie wykorzystywać ją wielokrotnie w projekcie.

Niechciane obiekty bądź grupy mogą być oczywiście usuwane z projektu. Opcja *Usuń* jest dostępna w menu tak jak wcześniej opisane funkcje edycji. Również klawisz *Delete* na klawiaturze powoduje usuwanie wybranych elementów.

4.1.6. Edytor właściwości obiektów



Rysunek 4.7. Edytor właściwości

Pokazane na powyższym rysunku okienko umożliwia szybką i wygodną edycję parametrów obiektów. Okienko można wywołać używając opcji *Właściwości* z menu kontekstowego prawego przycisku myszy lub z menu belki górnej. Dzięki temu edytorowi

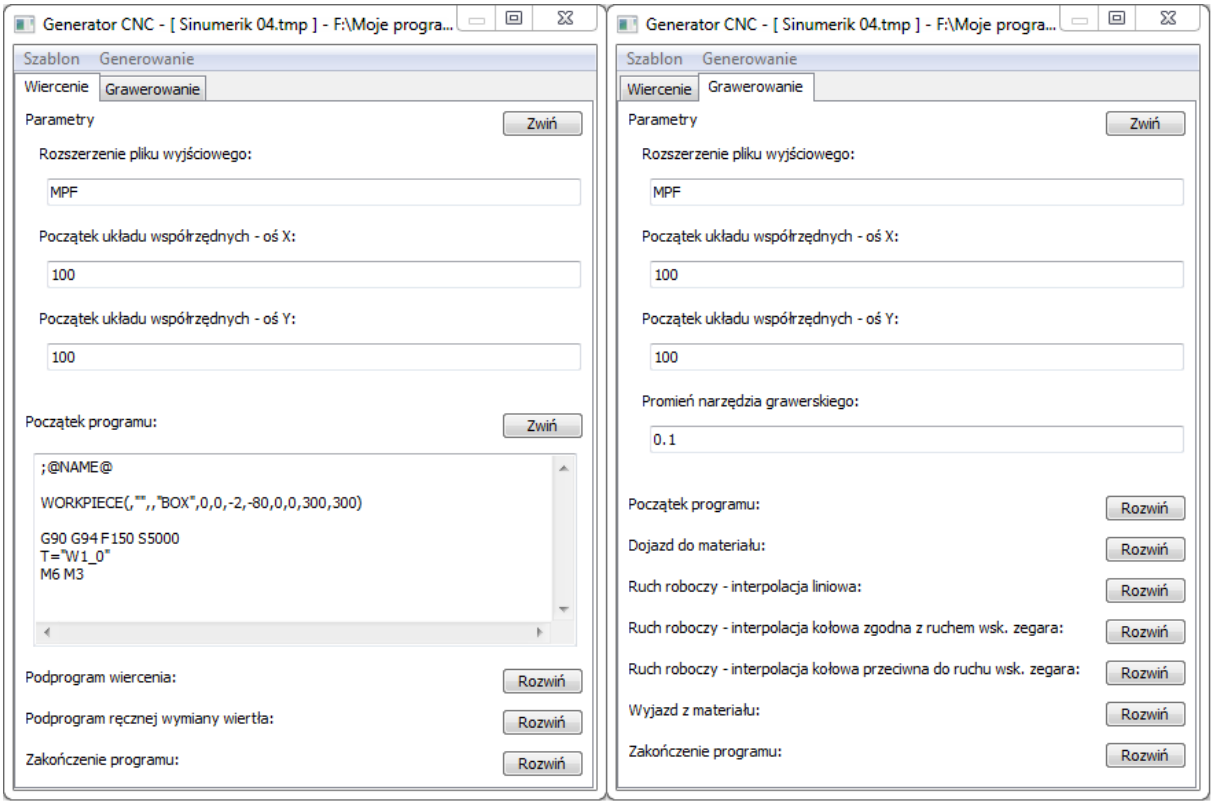
można zmieniać parametry wszystkich zaznaczonych elementów jednocześnie. Na rysunku 4.7. zaznaczone są trzy pola lutownicze i dwa odcinki ścieżki. Układ ten do swojego opisu potrzebuje trzech węzłów. Na okienku edytora widać jakie wymiary zewnętrzne oraz jakie średnice otworów aktualnie posiadają wybrane pola lutownicze. Kolejnym parametrem jest szerokość ścieżki. Oba zaznaczone odcinki ścieżki posiadają tą samą szerokość i jest ona widoczna na liście właściwości. Kolejną częścią okna jest lista węzłów używanych przez zaznaczone obiekty. Węzły posiadają swoje identyfikatory, które jednoznacznie je określają. Z pokazanej listy można wybrać interesujący nas węzeł i za pomocą dwóch ostatnich pól okna właściwości zmienić jego pozycję w modelu. Przesunięcie węzła na wybraną pozycję może być przydatne, gdy zachodzi potrzeba dokładnego zwymiarowania modelu z pominięciem siatki rysunkowej. W zaprezentowanym przykładzie występują wszystkie możliwe rodzaje elementów, których właściwości wyświetla edytor. Ponadto zarówno pola lutownicze jak i ścieżki mają te same parametry. W takim przypadku wszystkie pola są aktywne i wskazują wartość danej cechy obiektu. W przypadku, gdy wśród zaznaczonych obiektów brakuje obiektów posiadających pewną cechę, pole odpowiedzialne za jej edycję jest nie aktywne i wyświetla następującą treść: "---". Istnieje również możliwość, że zaznaczone obiekty posiadają cechę wspólną, lecz jej wartość nie jest taka sama dla wszystkich. Wówczas pole odpowiedzialne za edycje tej cechy jest aktywne, ale wskazuje wartość "---". Istnieje wówczas możliwość wpisania tam wartości liczbowej, czego wynikiem będzie ujednolicenie danej cechy wśród wszystkich zaznaczonych obiektów.

Okienko właściwości jest *zawsze na wierzchu*, oznacza to, że po jego wywołaniu będzie zawsze widoczne nad głównym oknem aplikacji, aż do momentu zamknięcia. Okienko może być oczywiście wielokrotnie zamykane i ponownie wywoływane w razie potrzeby. Takie podejście umożliwia podgląd i edycję parametrów *na bieżąco*. Lista parametrów zawsze odpowiada aktualnemu stanowi zaznaczenia obiektów.

4.2. Moduł CAM - funkcje i możliwości

Nieodłączną częścią pakietu *PCB CAM Processor* jest moduł CAM, umożliwiający generowanie programów obróbkowych celem wykonania wcześniej zaprojektowanego układu elektronicznego. Połączenie modułu CAD i CAM w integralną całość daję doskonałe narzędzie do prototypowania prostych układów w warunkach domowych czy akademickich.

Moduł CAM jest bardzo uniwersalny, umożliwia generowanie programów obróbkowych na niemal dowolny typ maszyny sterowanej numerycznie. Jest to możliwe dzięki zastosowaniu szablonów zgodnych z wybraną obrabiarką. Program obróbki obwodu drukowanego podzielony jest na dwie operacje: grawerowanie układu ścieżek oraz wiercenie otworów montażowych dla elementów elektronicznych. Każda z tych operacji została rozbita na szereg elementarnych zabiegów, które składają się na gotowy program. Użytkownik musi zdefiniować dwa szablony, po jednym dla każdej z operacji. Każdy szablon składa się z fragmentów kodu CNC zgodnych z językiem wybranej maszyny, zaś każdy fragment opisuje pewien elementarny zabieg. Każde pole szablonu posiada szczegółowy tekst podpowiedzi.

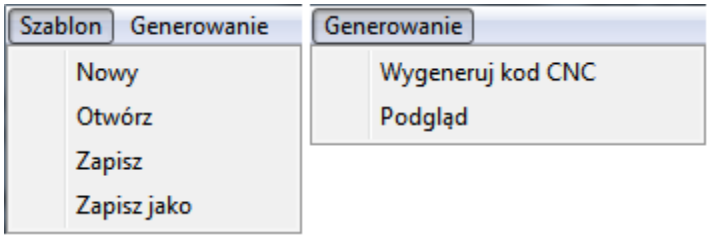


Rysunek 4.8. Szablony maszynowe

4.2.1. Edytor szablonów

Pokazane na rysunku 4.8. okienko edytora, pozwala na tworzenie i zarządzanie zapisanymi szablonami. Dysponując gotowym szablonem można jednym kliknięciem dostosować moduł CAM do współpracy z daną maszyną. Szablony są plikami zewnętrznymi więc można je przenosić między użytkownikami oprogramowania, co daje możliwość kooperacji i dzielenia się efektami swojej pracy. Aby wywołać edytor szablonów, należy użyć opcji *Generator CNC* z menu *Narzędzia* zlokalizowanego na górnej belce głównego okna aplikacji. Układ menu pokazano na rysunku 4.2.

Okienko edytora składa się z belki górnej oraz obszaru zawierającego ustawienia. Obszar ustawień zawiera dwie karty, po jednej dla operacji wiercenia i grawerowania. Dla zwiększenia czytelności poszczególne pola szablonu mogą być minimalizowane za pomocą przycisku *Zwiń/Rozwiń*. Belka górna zawiera menu *Szablon* z podstawowymi opcjami manipulacji na plikach oraz menu *Generowanie* służące do symulacji i generowania programów obróbkowych.



Rysunek 4.9. Menu belki górnej modułu CAM

4.2.2. Szablon *Wiercenie*

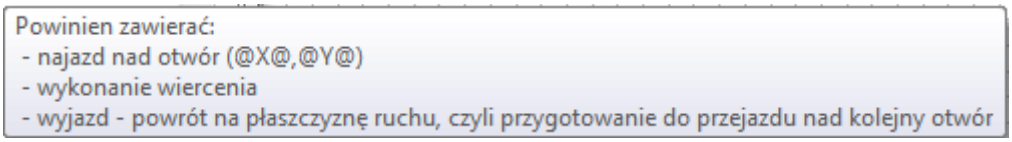
Jak pokazano na rysunku 4.8. każdy szablon składa się z listy parametrów takich jak rozszerzenie pliku wyjściowego czy przesunięcie punktu zerowego oraz listy fragmentów kodu obróbkowego. Operacja wiercenia została podzielona na następujące fragmenty:

- Początek programu obróbki

  Ta część powinna opisywać ustawienia początkowe programu, takie jak: wybór narzędzia, dobór parametrów skrawania itp. Zależnie od maszyny wymagane jest użycie różnych funkcji przygotowawczych. Ten fragment kodu jest oczywiście używany tylko jeden raz na początku programu.

- Podprogram wiercenia

  Opisuje kompletny cykl wymagany do wykonania pojedynczego otworu. Aby kolejne cykle mogły być wykonywane jeden po drugim, powinny być opracowane z zastosowaniem reguł jakie prezentuje tekst podpowiedzi dostępny dla tego pola szablonu. Podpowiedź dostępną po najechaniu myszą na pole pokazano poniżej.



Rysunek 4.10. Tekst podpowiedzi dla pola szablonu dot. wiercenia

- Podprogram wymiany wiertła

  W modelu obwodu drukowanego mogą występować otwory o różnej średnicy, dlatego zachodzi tutaj konieczność wymiany wiertła. Otwory wykonywane są w kolejności rosnącej. Po wykonaniu wszystkich otworów danej średnicy wywoływany jest podprogram wymiany narzędzia. W przypadku robota przemysłowego będzie on zawierał odpowiednie ustawienie się ramienia umożliwiające ręczną zmianę wiertła oraz zatrzymanie programu, aż do momentu wznowienia po zakończeniu czynności montażowych.

- Zakończenie programu

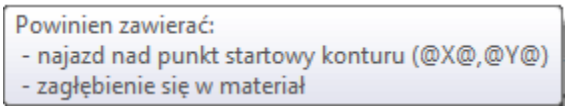
  W tym fragmencie powinny się znaleźć wszystkie czynności związane z poprawnym zakończeniem programu, takie jak odwołania niektórych funkcji przygotowawczych, wyłączenie obrotów wrzeciona czy wycofanie narzędzia na bezpieczną pozycję. Każdy język programowania maszyn w tym robotów przemysłowych posiada również pewną indywidualną składnię pliku. W tej sekcji powinny się zatem znaleźć wszystkie komendy zamykające plik programu.

Przedstawiony na rysunku 4.10. tekst podpowiedzi prezentuje format zapisu zmiennych modelu. Większość pól szablonu dysponuje pewną listą parametrów, która zawsze jest wypisana w tekście podpowiedzi. Dla wiercenia jest to pozycja otworu na płaszczyźnie opisana współrzędnymi X i Y. Omawiany fragment kodu jest wielokrotnie używany w programie obróbkowym dlatego musi być sparametryzowany. Pochodzące z modelu dane o środkach otworów są importowane do szablonu poprzez zapis *@nazwa parametru@*. Wystarczy zatem użyć w kodzie zapisu *@X@* czy *@Y@* aby poinformować moduł CAM, że w tym miejscu mają być podstawione współrzędne kolejnych otworów.

4.2.3. Szablon *Grawerowanie*

Szablon opisujący operacje grawerowania jest skonstruowany analogicznie jak poprzedni. W sekcji parametrów posiada dodatkowo deklaracje promienia narzędzia. Moduł CAM potrafi liczyć odsadzenie od konturu celem kompensacji promienia narzędzia. Jest to szczególnie istotne dla maszyn takich jak roboty przemysłowe nie posiadających tej funkcji w swoich układach numerycznych. Stosowania wbudowanej funkcji kompensacji dla maszyn CNC również niesie za sobą pewne problemy, dlatego łatwiejszą drogą jest wprowadzenie poprawki po stronie modułu CAM. Operacja grawerowania dzieli się na następujące zabiegi:

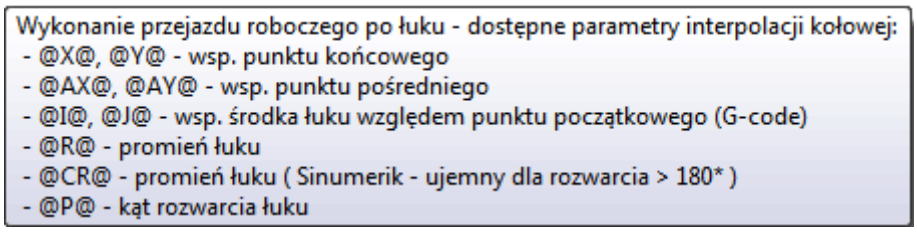
- Początek programu obróbki
  Ten fragment spełnia analogiczną funkcję jak w przypadku wiercenia.
- Dojazd do materiału
  Prezentowana operacja grawerowania polega na wykonywaniu zamkniętych konturów tworzących na płytce drukowanej pola o wspólnym potencjale elektrycznym, będące całkowicie odizolowane od pozostałej części obwodu drukowanego. Takie podejście w pierwszym kroku wymaga dojazdu narzędzia na początek konturu i zagłębienia się w materiał. Wspomnianą sekwencję omawia tekst podpowiedzi tego pola szablonu.



Rysunek 4.11. Tekst podpowiedzi dla pola szablonu dot. dojazdu do materiału

- Ruch roboczy – interpolacja liniowa
  Po zagłębieniu się w materiał należy prowadzić narzędzie po zadanym na podstawie modelu konturze. Narzędzie porusza się wówczas ruchem roboczym liniowym lub kołowym. Dla ruchy liniowego istnieją parametry *@X@* i *@Y@* będące współrzędnymi punktu docelowego.
- Ruch roboczy – interpolacja kołowa zgodnie z ruchem wskazówek zegara
  Opis interpolacji kołowej jest znacznie bardziej skomplikowany. Zależnie od języka programowania maszyny, istnieją różne sposoby opisu łuku. W G - kodzie istnieje rozróżnienie interpolacji ze względu na kierunek obróbki. Stąd w prezentowanym szablonie osobne pola dla ruchu zgodnego z ruchem wskazówek zegara, opisanego funkcją G2 oraz ruchu w przeciwnym kierunku, funkcja G3. W przypadku robotów przemysłowych nie ma takiego podziału, natomiast łuk opisywany jest poprzez punkt

pośredni. Istnieją również metody wykorzystujące promień, czy kąt rozwarcia łuku. W układach sterowania numerycznego *Sinumerik* firmy *Siemens* możliwe jest stosowanie zapisu łuku w oparciu o punkt końcowy i promień, który dodatkowo niesie informację o kącie rozwarcia łuku. Gdy kąt rozwarcia jest większy od 180° należy podać promień ze znakiem minus. Moduł CAM programu *PCB CAM Processor* wylicza szereg parametrów łuku dając możliwość elastycznego dostosowania do wybranej maszyny. Wszystkie dostępne parametry interpolacji kołowej pokazano na rysunku poniżej.



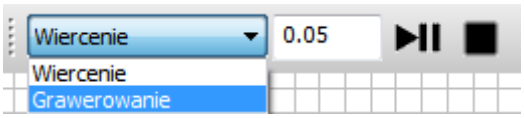
Rysunek 4.12. Tekst podpowiedzi dla pola szablonu dot. interpolacji kołowej

- Ruch roboczy – interpolacja kołowa przeciwnie do ruchu wskazówek zegara
  Opis analogiczny jak powyżej, takie same parametry interpolacji.
- Wyjazd z materiału
  Zakończeniem obróbki danego konturu musi być wyjazd z materiału na bezpieczną odległość, czyli przygotowanie do wykonania najazdu nad kolejny kontur.
- Zakończenie programu
  Ten fragment spełnia identyczną funkcję jak w przypadku wiercenia. Może być również użyty do połączenie programu grawerowania i wiercenia w całość poprzez wywołanie odpowiednich podprogramów. Oczywiście kolejność wykonania operacji grawerowania i wiercenia jest dowolna i zależy od upodobań programisty.

4.2.4 Symulacja przebiegu obróbki

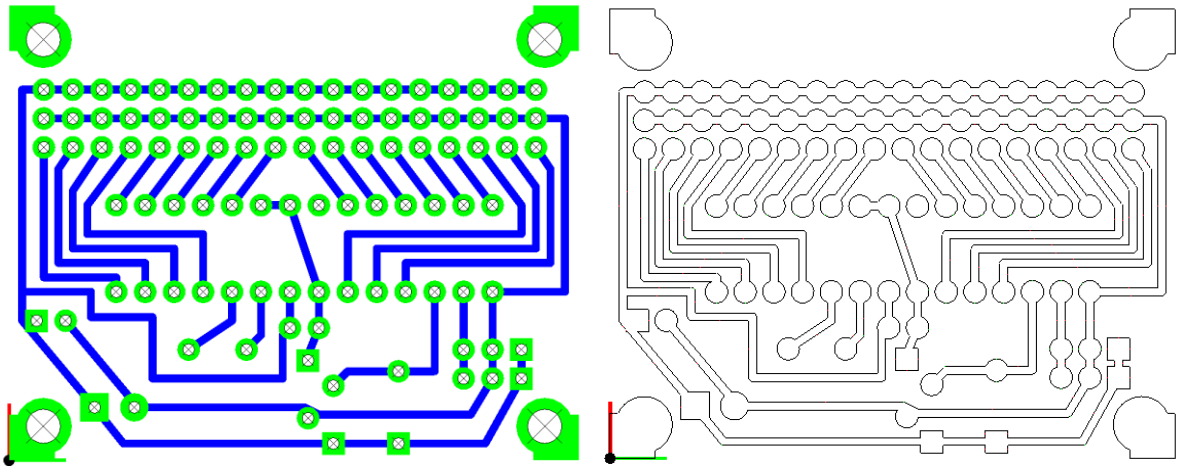
Ważnym etapem wdrażania obróbki przy użyciu systemu CAM jest weryfikacja poprawności wygenerowanej ścieżki narzędzia. Prezentowane oprogramowanie posiada wbudowany symulator zarówno operacji wiercenia jak i grawerowania. Symulacja prowadzona jest w grafice 2D, co w zupełności wystarcza do celów weryfikacji ścieżki.

Symulacja wyświetlana jest w głównym oknie aplikacji, a do sterowania jej przebiegiem służą opcje zlokalizowane po prawej stronie belki rysunkowej (rysunek 4.1.). Poniższy rysunek przedstawia wygląd belki symulacji.



Rysunek 4.13. Belka symulacji – wybór operacji

Aby wejść w tryb symulacji należy wywołać okno edytora szablonów pełniące w tym przypadku dodatkowo rolę menedżera modułu CAM. Następnie z belki górnej tego okna aktywować opcję *Podgląd* z menu *Generowanie*. Układ menu pokazano na rysunku 4.9. Wyjście z trybu symulacji odbywa się poprzez odznaczenie opcji *Podgląd* lub poprzez wciśnięcie klawisza *Esc* na klawiaturze. Podczas symulacji, opcje zlokalizowane na belce rysunkowej są nieaktywne, nie ma możliwości edycji modelu. Aktywna jest w tym momencie belka symulacji pokazana na rysunku 4.13. Z uwagi na podział obróbki na dwie operację istnieje możliwość wyboru, której z operacji aktualnie dotyczy symulacja. Kolejne pole na belce definiuje posuw dla operacji grawerowania podawany w [m/s]. Ostatnie dwa przyciski służą do sterowania przebiegiem symulacji. Pierwszy z nich uruchamia, zatrzymuje i wznawia symulację, a drugi zatrzymuję ją i cofa do początku. Funkcje te są analogiczne jak w przypadku odtwarzaczy muzyki czy filmów. Poniższy rysunek przedstawia przykładowy model oraz efekt pracy modułu CAM. Pokazanie pracy symulatora w pisemnej publikacji jest oczywiście nie możliwe.



Rysunek 4.14. Przykładowy model oraz jego obliczony kontur – przygotowanie symulacji grawerowania

4.2.5 Generowanie kodu CNC

Ostatnim etapem opracowywania programów przy pomocy systemów CAM jest wygenerowanie gotowego programu obróbkowego. Program ten powinien być tak przygotowany, aby można go było bezpośrednio wgrać do maszyny i uruchomić. Moduł CAM prezentowanej aplikacji umożliwia pełne dostosowanie do wymogów jakie stawia układ sterowania wybranej maszyny, poprzez zastosowanie opisanych powyżej szablonów maszynowych. Wygenerowanie programów sterujących odbywa się poprzez użycie opcji *Wygeneruj kod CNC* z menu *Generowanie* górnej belki okna modułu CAM. Układ menu pokazano na rysunku 4.9. Efektem działania wspomnianej funkcji jest powstanie dwóch plików tekstowych o rozszerzeniach zdefiniowanych przez szablony. Utworzone pliki są automatycznie otwierane w programie *Notatnik* dla ostatecznej weryfikacji poprawności działania aplikacji *PCB CAM Processor*. Użytkownikowi pozostaje teraz przenieść pliki na przygotowane wcześniej stanowisko i uruchomić programy obróbkowe.

5. BUDOWA WEWNĘTRZNA APLIKACJI

5.1. Podstawowe struktury danych

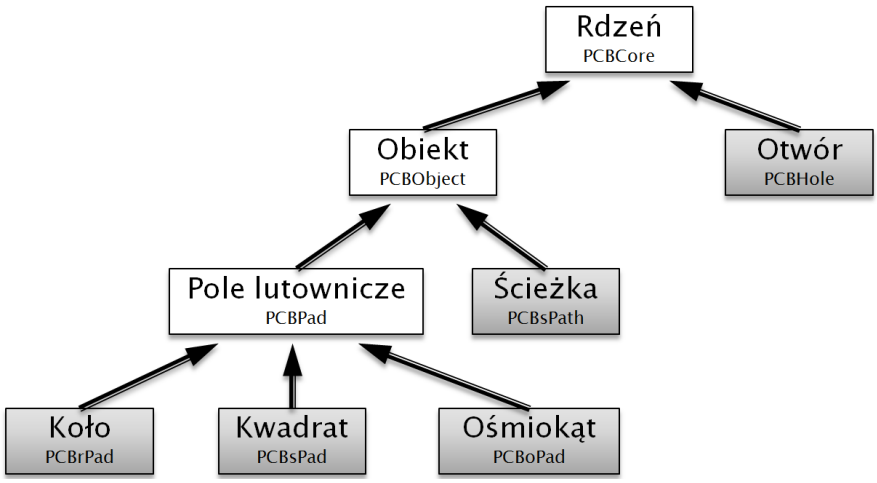
Aplikacja *PCB CAM Processor* jest w całości napisana obiektowo w języku C++. Obiektowe podejście do tworzenia aplikacji jest szczególnie istotne przy tworzeniu dużych i złożonych projektów programistycznych.

W poprzednim rozdziale pokazano, że model projektowany przy użyciu modułu CAD składa się z wielu różnych elementów, które posiadają pewne cechy. Niektóre z tych elementów są do siebie w pewnym sensie podobne tak jak np. pola lutownicze lub całkowicie różne jak ścieżka i otwór. Podstawowymi obiektami widocznymi dla użytkownika są pola lutownicze w trzech możliwych kształtach, fragmenty ścieżek oraz otwory. Wymienione elementy, w myśl programowania obiektowego należą do różnych *klas*. Klasy stanowią pewnego rodzaju schemat budowy według którego powstają kolejne obiekty. Może istnieć dowolna liczba obiektów danej klasy. Wszystkie są do siebie podobne, posiadają takie same cechy czy *umiejętności*, różnią się natomiast wartością danej cechy.

Tworzenie aplikacji typu CAD wymaga pewnego podejścia umożliwiającego opis rzeczywistego świata w przestrzeni wirtualnej komputera. Program komputerowy musi w pewnym sensie odzwierciedlać świat rzeczywisty mimo operowania na zupełnie innym poziomie abstrakcji. Z jednej strony mamy do czynienia z prawdziwymi obiektami posiadającymi dane wymiary czy kształty, a z drugiej kod programu próbujący opisać to wszystko w zupełnie innej rzeczywistości. Programowanie zorientowane obiektowo umożliwia jednak pewne zbliżenie do siebie tych dwóch zupełnie różnych światów poprzez stosowania przeniesienia świata rzeczywistego do wirtualnej rzeczywistości niemal w skali jeden do jeden. Zarówno obiekty rzeczywiste jak i ich wzajemne zależności mogą być opisane przy użyciu języka zbliżonego do naturalnego języka człowieka, dając bardzo przejrzysty i zrozumiały opis.

Jednym z mechanizmów programowania obiektowego, który realizuje wspomniane przenoszenie rzeczywistości do programu komputerowego jest *dziedziczenie*. Umożliwia ono tworzenie nowych klas na podstawie już istniejących, dodając nowe pola czy metody klasy. Klasa pochodna jest zupełnie nowym typem, ale wciąż można korzystać z obiektów tej klasy jak gdyby należały do klasy bazowej. Taki zabieg jest możliwy stosując rzutowanie typów, które w tym kierunku hierarchii dziedziczenia jest czymś naturalnym dla kompilatora. Istotę

i zastosowanie dziedziczenia obrazuje przykład systematyki organizmów w biologii. Biorąc pod uwagę dobrze znane gatunki jak: pies, wilk, lew czy tygrys możemy powiedzieć o nich następujące fakty. Każdy gatunek to pewnego rodzaju klasa, a każdy przedstawiciel tego gatunku to obiekt danej klasy. Każdy z przedstawicieli tych gatunków jest ssakiem, czyli klasa *ssak* jest klasą bazową dla klas *psowate* i *kotowate*. Rozpatrując biologiczny podział gatunków na rzędy, rodziny itd. zauważalne jest podobieństwo w stosunku do hierarchii dziedziczenia klas w programowaniu obiektowym.



Rysunek 5.1. Schemat dziedziczenia klas stosowanych do opisu modelu CAD[7]

Analizując dalej, przedstawiony powyżej przykład, można powiedzieć, że nie mogą istnieć zwierzęta klasy *ssak* czy klasy *psowate* i *kotowate*. Klasa w rozumieniu biologicznym jest traktowana w tym przykładzie jako gatunek, zatem nie istnieje gatunek *ssak*. Kolejnym istotnym pojęciem z zakresu programowania obiektowego doskonale opisującym ten przypadek jest *klasa abstrakcyjna*. Jest to taka klasa która nie może mieć swoich przedstawicieli w postaci obiektów. Abstrakcyjna jest każda klasa która posiada choć jedną metodę *czysto wirtualną*. Takie metody muszą być zdefiniowane w klasie pochodnej po klasie abstrakcyjnej. Tylko wówczas nowa klasa nie jest już abstrakcyjna i można tworzyć obiekty tej klasy. Klasy abstrakcyjne są sposobem na ujednolicenie pod względem funkcjonalności obiektów różnych klas pochodnych.

Na rysunku 5.1. pokazano schemat dziedziczenia klas stosowanych do opisu modelu budowanego w aplikacji. Klasy zaznaczone na biało, czyli *PCBCore*, *PCBObject* oraz *PCBPad* są abstrakcyjne natomiast pozostałe klasy zaznaczone na szaro stanowią reprezentacje rzeczywistych elementów płyty PCB, widocznych na ekranie. Dzięki zastosowaniu takiego podziału możliwe jest wspomniane ujednolicenie. Klasa *PCBCore*, stanowiąca

pewnego rodzaju rdzeń dla wszystkich innych klas, posiada np. wirtualną metodę *Draw()*. Każda z rzeczywistych klas dziedzicząc po klasie *PCBCore* nadpisuję tą metodę własną implementacją funkcji rysowania. Istotnie każdy z elementów widocznych na ekranie wymaga narysowania, ale zależnie od jego interpretacji graficznej sposób rysowania różni się. Ujednolicenie obiektów polega w tym przypadku na tym, że wystarczy posiadać tylko jeden kontener w postaci np. tablicy, listy jedno- lub dwukierunkowej zawierającej wskaźniki typu bazowego na wszystkie obiekty typów pochodnych, aby z powodzeniem narysować je wszystkie. Metody wirtualne polegają bowiem na tym, że wywołując taką metodę na rzecz obiektu pochodnego, ale przy użyciu wskaźnika typu bazowego, wykonywana jest nadpisana metoda klasy pochodnej.

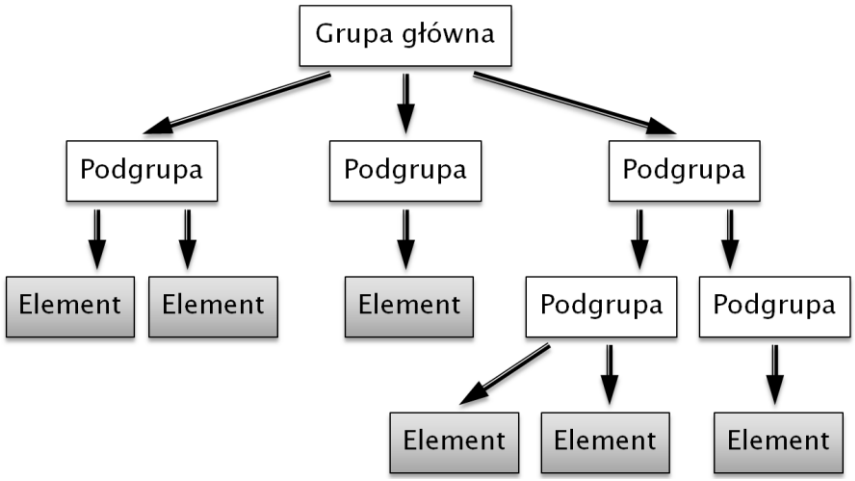
Powyższy rozważania są próbą przybliżenia tematyki związanej z programowaniem obiektowym w przystępny i zrozumiały sposób. Nie stanowi to ścisłego opisu w sensie teorii programowania, gdyż nie jest to przedmiotem tej pracy.

Obiektami spajającymi elementy oraz nadającymi wymiary modelom są węzły. Nie posiadają one reprezentacji graficznej, ale są edytowalne przez użytkownika aplikacji poprzez opisane w poprzednim rozdziale okno właściwości. Każdy element płyty PCB widoczny na ekranie potrzebuje do swojego opisu przynajmniej jednego węzła. W sensie programowania, obiekty klas zaznaczonych na szaro na rysunku 5.1. posiadają pola zapamiętujące wskaźniki na obiekty typu węzeł. Operacje przesuwania obiektów wykonywane są tak naprawdę na węzłach, bo to one są pewnego rodzaju nośnikami wymiarów. Dzięki temu, że kilka elementów może korzystać z tego samego węzła, możliwe jest powiązanie tych elementów. Efektem tego powiązania jest to, że przesunięcie jednego węzła powoduje zmianę położenia wszystkich elementów korzystających z jego pozycji. Węzły są zupełnie osobną klasą obiektów i dlatego nie występują na schemacie przedstawionym na rysunku 5.1. Stanowią jednak podstawę opisu przestrzennego wszystkich obiektów widocznych na tym rysunku.

5.2. Grupy elementów

W rozdziale poświęconym możliwościom i obsłudze aplikacji pokazano opcję grupowania obiektów. Realizacja takiego podejścia wymagała wprowadzenia nowej klasy stanowiącej kontener umożliwiający zapamiętanie wskaźników na obiekty typu *PCBCore* oraz wskaźników na inne grupy stanowiące w takim układzie podgrupy. Po utworzeniu nowego projektu lub po starcie aplikacji istnieje już jeden obiekt klasy *Group*, który jest grupą główną i pełni rolę głównego kontenera tzw. *GroupManagera*. Najprostszą grupą

dodawaną przez użytkownika jest pole lutownicze składające się z pola kształtowego klasy *PCBrPad, PCBsPad* lub *PCBoPad* oraz otworu klasy PCBHole. Połączenie tych dwóch elementów w grupę jest niezbędne do prawidłowego funkcjonowania modelu. Użytkownik aplikacji w danej chwili ma dostęp do poszczególnych podgrup grupy głównej. Poprzez kliknięcie myszą zaznacza obiekty widoczne na ekranie. Gdyby otwór i pole kształtowe były widoczne dla użytkownika jako osobne elementy mógłby je niezależnie przemieszczać, co nie miałoby sensu w myśl projektowania płyt PCB.



Rysunek 5.2. Schemat zapisu elementów z podziałem na grupy[7]

Wszystkie operacje prowadzone przez użytkownika dotyczą podgrup grupy głównej. Stąd dla zachowania porządku, istnienie grup jednoelementowych takich jak grupa zawierająca fragment ścieżki. Takie podejście umożliwia łatwe przeprowadzenie grupowania elementów, a właściwie grup. Operacje grupowania/rozgrupowania również odbywają się wyłącznie na pierwszym poziomie drzewka przedstawionego na rysunku 5.2. Operacja grupowania polega na utworzeniu nowej grupy i zapisaniu na jej liście, wskaźników na grupowane grupy. Jednocześnie wskaźniki na owe grupy są usuwane z listy należącej do grupy głównej, a nowo powstała grupa jest na niej zapisywana. Obrazuje to częściowo rysunek 5.2. Ostatnia podgrupa grupy głównej stanowi zgrupowanie dwóch poprzednich podgrup. Oczywiście grupa główna powinna teraz posiadać tylko jednego członka ostatnią z widocznych podgrup.

Rozgrupowanie jest operacją odwrotną, podlegają mu jedynie grupy posiadające jakieś podgrupy. W tej operacji posiadane przez grupę wskaźniki są przekazywane grupie głównej, a ona sama jest niszczona. Nie da się rozgrupować układu pola lutowniczego, gdyż jest to grupa wbudowana, posiadająca tylko elementy, nie posiadająca żadnych podgrup.

Zastosowanie opisu widocznego na rysunku 5.2. powoduje że wszystkie prowadzone operacja mają charakter rekurencyjny. Najbardziej podstawową operacją jest zaznaczenie. Każda grupa oraz każdy jej element posiada pole zapamiętujące stan zaznaczenia. Kliknięcie na ekranie wyzwala funkcję sprawdzającą czy dany punkt znajduje się wewnątrz któregokolwiek elementu grupy. Jeśli elementy grupy nie spełniają kryteriów sprawdzanie jest kontynuowane w głąb drzewka. Ta sama funkcja wykonywana jest na rzecz podgrup. Znalezienie elementu, na który wskazuje kursor myszy, powoduje zatrzymanie przeszukiwania. Dana grupa, w której znajduje się wskazany element musi zostać zaznaczona. Ustawiany jest znacznik dla wszystkich elementów tej grupy, następnie funkcja wykonywana jest na rzecz podgrup. Kolejno, na zasadzie rekurencji, dana funkcja przechodzi w dół drzewka, aż do grup nie posiadających już żadnych podgrup. Takie podejście umożliwia budowanie drzewka o dowolnej liczbie poziomów czy rozgałęzień stosując jeden prosty algorytm.

Rekurencyjne podejście jest stosowane również w operacjach usuwania, duplikowania, czy wreszcie zapisu i odczytu pliku, co będzie szczegółowo opisane później. Duplikowanie jest szczególnie ciekawą operacją. Wymaga skopiowania nie tylko wszystkich elementów z zachowaniem układu grup, co załatwia opisane wyżej podejście, ale również kopiowania węzłów. Z uwagi na fakt, że każdy węzeł może być wykorzystywany przez wiele obiektów, duplikacji muszą również podlegać wiązania między węzłami, a obiektami. W prezentowanej aplikacji zostało to rozwiązane w następujący sposób. Każdy element posiada pewną liczbę wskaźników na węzły, z których korzysta. Posiada również metodę umożliwiającą wykonanie kopi samego siebie. Z uwagi na fakt, że wiele obiektów korzysta z tego samego węzła, nie podlegają one tutaj bezpośredniemu kopiowaniu. Gdyby oba elementy wykonały kopię swoich węzłów, każdy używałby swojej kopii i nie byłoby już powiązania między tymi elementami. Ponadto do celów zapisu projektu do pliku, każdy węzeł posiada unikatowe ID, którego nadawanie również stanowiłoby w tym przypadku problem. Należało zatem wprowadzić pewien zewnętrzny system zarządzania węzłami. Utworzono nową klasę o nazwie *NodeManager*, zawierającą kontener obiektów typu *PCBNode*, licznik umożliwiający nadawanie kolejnych ID, funkcje zapisu i odczytu pliku oraz szereg innych funkcji potrzebnych do manipulacji węzłami. *NodeManager* umożliwia wykonanie kopii węzła w przesunięciu o wektor oraz zapisanie w nim informacji o ID węzła, z którego powstał. Funkcja ta jest używana przed duplikacją elementów. W efekcie, powstaje zbiór węzłów, będących kopią zaznaczonego obszaru, oraz każdy ze skopiowanych węzłów

posiada informację, który węzeł był jego pierwowzorem. W tej chwili możliwe jest przeprowadzenie duplikacji obiektów. Każdy obiekt, chcąc utworzyć swoją kopię, musi uzyskać od *NodeManagera* wskaźniki na kopie używanych przez siebie węzłów. Tylko w przypadku uzyskania pozytywnej odpowiedzi może użyć tych węzłów i podłączyć je do tworzonej kopii samego siebie. Widoczny na ekranie podgląd operacji duplikacji jest generowany znacznie prościej. Kontener główny obiektów umożliwia dodatkowe rysowanie zaznaczonych elementów przesuniętych o wektor względem pierwotnego położenia. Nie wymaga to tworzenia węzłów ani obiektów, dlatego jest proste w anulowaniu. Dopiero moment wstawienia zapoczątkowuje opisany wyżej scenariusz tworzenia nowych węzłów i elementów.

Opisany mechanizm wydaje się być skomplikowany, jednak zasada jego działania jest prosta. Istotnie pod względem programistycznym stanowi to pewne wyzwanie i wymaga mocnych umiejętności z zakresu programowania obiektowego. Z uwagi na to, w niniejszej pracy nie przedstawiono kodów źródłowych, ani szczegółowych algorytmów. Powyższe opisy stanowią jedynie bardzo ogólny opis niektórych problemów koniecznych do rozwiązania przy budowie tej klasy oprogramowania.

5.3. Budowa pliku projektu

Plik projektu posiada składnię typu XML czyli języka znacznikowego stosowanego przy tworzeniu stron internetowych w języku HTML. Umożliwia to łatwy sposób zapisu drzewka grup. Plik podzielony jest na trzy sekcje:

- Sekcja ustawień

```
<settings>
      <viewpoint>-0.709519, -1.334059</viewpoint>
      <scale>45.949730</scale>
</settings>
```

  Tutaj zapisane są ustawienia okna edycji. Po wczytaniu pliku, rozmiar i położenie modelu będą identyczne jak przy zapisie. Stanowi to pewną analogie to innych systemów CAD.

- Sekcja węzłów

```
<nodes>
      <node>n|0|2.540000|2.540000</node>
      <node>n|1|5.080000|2.540000</node>
</nodes>
```

Każdy węzeł opisany jest kolejno ID oraz współrzędnymi X i Y w modelu.

- Sekcja grup i elementów

```
<group>
      <elements>
      </elements>
      <groups>
            <group>
                  <elements>
                        <element>1|1.800000|0</element>
                        <element>h|1.000000|0</element>
                  </elements>
                  <groups>
                  </groups>
            </group>
            <group>
                  <elements>
                        <element>2|1.800000|1</element>
                        <element>h|1.000000|1</element>
                  </elements>
                  <groups>
                  </groups>
            </group>
            <group>
                  <elements>
                        <element>4|0.400000|0|1</element>
                  </elements>
                  <groups>
                  </groups>
            </group>
      </groups>
</group>
```

Pomiędzy zewnętrznymi znacznikami *<group>…</group>* znajduje się opis grupy głównej, a w niej lista elementów oraz podgrup. Jak widać lista elementów jest pusta. Natomiast pomiędzy *<groups>…</groups>* znajdują się trzy tagi *<group> …</group>* zawierające podobnie jak grupa główna, listę elementów oraz listę podgrup, która w tym przypadku jest pusta. Jest to przykład pliku opisującego układ złożony z dwóch pól lutowniczych połączonych ścieżką. Element *h* to otwór o podanej średnicy i węźle, *1* oznacza pole w kształcie koła, *2* w kształcie kwadratu, *3* w kształcie ośmiokąta. Kolejny parametr to wymiar zewnętrzny oraz ID używanego węzła. Element *4* to odcinek ścieżki o danej dalej szerokości oraz numerach ID wykorzystywanych węzłów.

Z uwagi na prosty i przejrzysty zapis w postaci pliku tekstowego istnieje możliwość ręcznej edycji pliku oraz weryfikacji jego poprawności. Taka konstrukcja pliku umożliwia zapis drzewka grup o dowolnej liczbie poziomów.

Zapis i odczyt pliku wykonywany jest przy użyciu managera węzłów oraz managera grup. Odczyt rozpoczyna się od utworzenia przez *NodeManagera* węzłów oraz nadaniu im ID zgodnie z tą sekcją pliku. Następnie plik jest czytany przez *GroupManegera* celem utworzenia kolejnych podgrup i obiektów. Ponownie jest tutaj użyty mechanizm rekurencji. Najpierw tworzone są podgrupy grupy głównej, następnie fragmenty opisujące je są im przekazywane i proces tworzenia jest rozwijany w dół drzewka. Każda grupa na podstawie sekcji elementów tworzy kolejne elementy. Jest to możliwe dzięki funkcją będącym konstruktorami obiektów, przyjmującym za argument tekst znajdujący się pliku. Przed utworzeniem elementu sprawdzane jest również, czy istnieją wymagane węzły. To znaczy, czy na poprzednim etapie wczytywania pliku nie wystąpiły błędy, przez co mogły nie zostać utworzone wszystkie opisane węzły. Jeśli wszystko jest w porządku, element zostaje utworzony i zapisany na liście elementów danej grupy. *NodeManager* jest tutaj dystrybutorem wskaźników na węzły o podanym ID.

Zapis do pliku odbywa się podobnie. Również tu wykorzystywane są *Managery*. Najpierw do pliku dodawana jest sekcja ustawień widoku. Później *NodeManager* tworzy sekcje węzłów, a na koniec *GroupManager* dodaje sekcję elementów. Na przykładzie operacji zapisu można łatwo przedstawić schemat rekurencyjnego postępowania w drzewku. Grupa główna otwiera plik i zapisuje tam znaczniki otwierające, następnie przekazuje tworzenie pliku kolejno swoim podgrupą. Każda z nich zapisuje tam swoje elementy lub otwiera sekcje podgrup i przekazuje dalej. Kiedy wszystkie podgrupy domkną daną sekcję obiekt nadrzędny również może to zrobić. Finalnie, grupa główna zamyka sekcję pliku poświęconą elementom.

6. PROBLEMATYKA OBLICZEŃ

6.1. Obliczenia zmiennoprzecinkowe

W technologii komputerowej występują zasadniczo dwa podstawowe rodzaje zmiennych liczbowych: całkowite oraz zmiennoprzecinkowe. W obrębie tych dwóch grup istnieją oczywiście zmienne różniące się co do zajmowanej ilości pamięci, a co za tym idzie różniące się możliwościami zapisu liczb. W przypadku liczb całkowitych istnieją typy zajmujące 1, 2, 4, a nawet 8 bajtów. Można oczywiście spotkać się ze specjalnymi implementacjami wielobajtowymi służącymi do precyzyjnych obliczeń na bardzo dużych liczbach całkowitych. Typy zmiennoprzecinkowe służą z kolei do zapisu liczb rzeczywistych. Ten typ liczbowy pozwala zapisać dowolnie duże czy małe liczby, ponieważ zapis jest tutaj podobny do notacji wykładniczej. Długość zmiennej określa w tym przypadku dokładność zapisu. Podobnie jak w notacji, im więcej cyfr znaczących tym dokładniejszy zapis. Niezależnie od rozmiaru zmiennej, zawsze występuje błąd zaokrąglenia, który jest w tych rozważaniach kluczowym problemem.

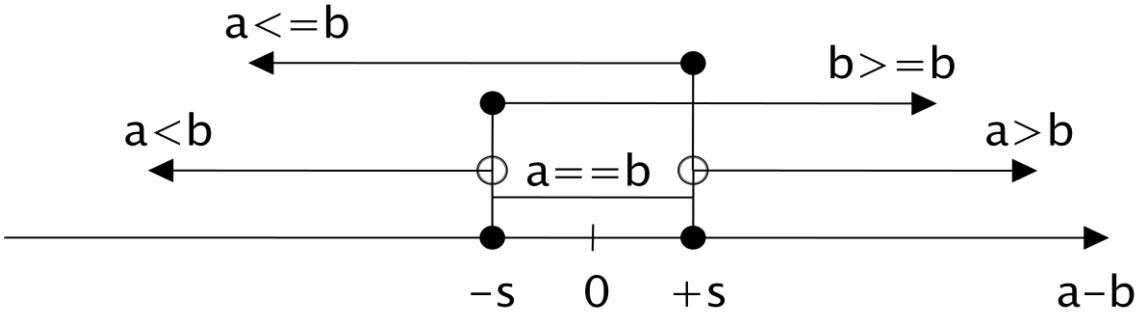
$$\begin{aligned} a &= (x + y) + z \\ b &= x + (y + z) \\ \boldsymbol{a} &\boldsymbol{\neq b} \end{aligned} \tag{1}$$

Zaprezentowany powyżej przykład obliczeniowy obrazuje jeden z ważniejszych problemów w arytmetyce liczb zmiennoprzecinkowych. Widać tutaj, że kolejność obliczeń może mieć znaczenie dla końcowego wyniku. Liczby a i b są oczywiście równe w sensie matematycznym, ale w pamięci komputera będą się nieznacznie różnić.

Prowadząc obliczenia komputerowe należy pamiętać o istnieniu zaprezentowanego wyżej problemu. Wyliczenie tej samej wartości dwoma różnymi sposobami nie da nigdy identycznego wyniku. Nie można zatem bezpośrednio porównywać tych liczb. Twórcy różnego rodzaju algorytmów często zapominają o tym, pisząc warunki równościowe. Sprawdzenie w programie komputerowym równości tych dwóch pozornie równych liczb nie jest takie oczywiste z uwagi na wspomniane błędy zaokrągleń.

Problem ten można łatwo rozwiązać badając różnicę liczb. Jeśli różnica dwóch liczb jest dostatecznie mała oznacza to, że można je uznać za równe. Przyjmując zatem pewnego rodzaju skalę podobieństwa S, można powiedzieć że dwie liczby są sobie równe gdy wartość bezwzględna ich różnicy jest mniejsza lub równa S. Stosując tą metodę dalej, można

wyprowadzić zależności umożliwiające skonstruowanie pozostałych operatorów: mniejszy, większy, mniejszy lub równy oraz większy lub równy. Przykładowo jeśli a < b oznacza to, że a - b musi być mniejsze od -S. Wartość skali podobieństwa można dobierać dowolnie, zależnie od wymaganej dokładności operatorów porównania. Obliczenia prowadzone w oprogramowaniu stosują zamiennie S na poziomie 10^{-9}, 10^{-6} oraz 10^{-3}. Zmiana S jest czasem wymagana aby ustalić pewną hierarchię warunków porównania. Poniżej przedstawiono interpretację graficzną opisanej metody porównywania.



Rysunek 6.1. Operatory porównania z zastosowaniem skali podobieństwa

Na osi poziomej zaznaczono różnicę liczb a i b. Rysunek przedstawia odniesienie wartości tej różnicy do wzajemnej relacji między liczbami a i b zapisanymi za pomocą operatów wykorzystywanych we współczesnych językach programowania. Można powiedzieć, że przedział <-s, s> jest pewnym bliskim otoczeniem punktu 0. Metoda ta zastępuje pojęcie zera jako punktu, które nie istnieje w prezentowanej arytmetyce, zerem traktowanym jako przedział, uwzględniającym błędy zaokrągleń. Zaprezentowana metoda jest łatwa do implementacji w języku C++ przy użyciu makr przedstawionych poniżej.

```
#define EQ_DOUBLE(a,b,s) ( abs(a-b) <=  s )
#define LT_DOUBLE(a,b,s) (    (a-b) <  -s )
#define LE_DOUBLE(a,b,s) (    (a-b) <=  s )
#define GT_DOUBLE(a,b,s) (    (a-b) >   s )
#define GE_DOUBLE(a,b,s) (    (a-b) >= -s )
```

Przedstawione powyżej zapisy reprezentują kolejno operatory: =, <, ≤, > oraz ≥. Są to makra, których wartość logiczna jest zależna od podanych wartości a i b oraz skali podobieństwa. Użycie ich w programie jest bardzo proste co pokazano na poniższym przykładzie.

```
if( EQ_DOUBLE( a, b, 0.001 ) )
{
     ...
}
```

Przykład prezentuje użycie makra do porównania dwóch liczb z uwzględnieniem skali podobieństwa 0.001. Wewnątrz warunku powinny się teraz znaleźć instrukcje mające się wykonać w przypadku równości liczb a i b.

6.2. Prymitywy

Obliczenia prowadzone w aplikacji służą wyznaczeniu wspólnego konturu elementów modelu. Każdy element dziedziczący po klasie *PCBObject* (rysunek 5.1.) posiada pole ograniczone pewnym zbiorem linii i łuków stanowiących jego kontur. Prymitywy są obiektami reprezentującymi wspomniane linie i łuki.

Klasa *Primitive* umożliwia zapisanie prymitywu liniowego lub łukowego, ponieważ posiada pole decydujące o typie. Takie podejście jest łatwiejsze w implementacji i z uwagi na to, że są tylko dwa możliwe typy nie został tu wprowadzony mechanizm dziedziczenia i ujednolicenia z użyciem klas abstrakcyjnych opisany w poprzednim rozdziale.

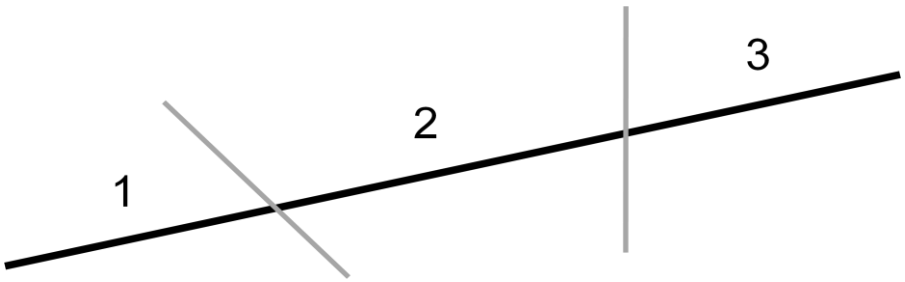
Podstawową funkcjonalnością prymitywów jest obliczanie punktów przecięć z innymi prymitywami. Klasa posiada również wiele metod pomocniczych umożliwiających wyznaczanie środka, sprawdzanie czy punkt należy do prymitywu czy odwracanie kierunku. W dalszych rozważaniach istotne będą również metody umożliwiające sortowanie punktów wzdłuż prymitywu, podział na fragmenty, czy łączenie współliniowych prymitywów. Jak widać mimo sugerującej nazwy tej klasy, posiada ona wiele funkcjonalności, z których nie wszystkie zostały tu wymienione.

Nazwa klasy pochodzi od typu danych jakie opisuje. Linia i łuk są bowiem bardzo prostymi formami geometrycznymi. Słowo prymityw jest często stosowane w nomenklaturze grafiki komputerowej, np. w bibliotekach graficznych OpenGL. W grafice występują takie elementarne formy jak: linie, łańcuchy linii, trójkąty, wielokąty czy wreszcie elementy mogące opisać jakąś powierzchnię np. za pomocą ciągu trójkątów czy czworokątów. Nie istnieją tu jednak żadne formy typu okrąg ani kula, gdyż ich przetwarzanie byłoby dla procesorów graficznych bardzo trudne z uwagi na ciągły i trygonometryczny charakter. Łatwiej jest prowadzić transformacje graficzne na zbiorach linii przybliżających te formy. W prezentowanej aplikacji zastosowano łuki z uwagi na ich zastosowanie w opisie modeli oraz odzwierciedlenie w układach sterowania numerycznego. Przejazdy w interpolacji

kołowej są możliwe w układach CNC, dlatego łatwiej i dokładniej można opisać je łukiem niż zbiorem linii prostych.

6.2.1 Sposób opisu linii

Linia do swojego opisu wymaga jedynie dwóch punktów które można interpretować jako początek i koniec. W przypadku linii kolejność tych punktów nie ma wpływu na opis. Prymitywy posiadają listę punktów należących do nich. Początkowo, nowo utworzony obiekt typu linia zawiera tylko dwa punkty. Dodatkowe punkty powstające na skutek przecięć z innymi prymitywami są umieszczane na wspomnianej liście. Lista punktów podlega sortowaniu wzdłuż kierunku prymitywu od punktu początkowego do końcowego. W efekcie sortowania powstaje pewien zbiór, w którym każde dwa kolejne punkty tworzą linię stanowiącą fragment pierwotnej.



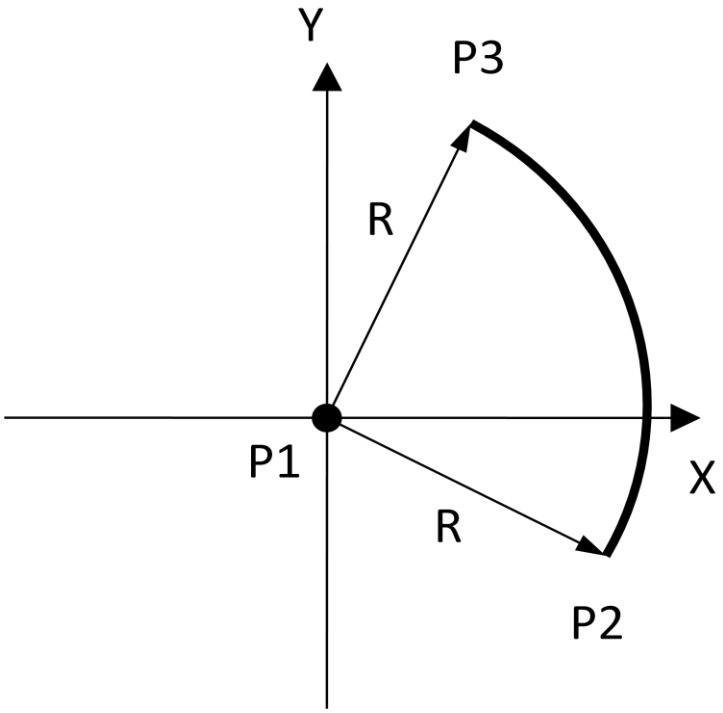
Rysunek 6.2. Przecinanie się prymitywów oraz podział na fragmenty[7]

Na powyższym rysunku pokazano podział prymitywu na fragmenty oznaczone 1, 2, 3 po obliczeniu punktów przecięć z dwoma innymi prymitywami. Sortowanie punktów gwarantuje, ze kolejne pary punktów tworzą rozłączne fragmenty, następujące po sobie.

6.2.2 Sposób opisu łuku

Jak pokazano w rozdziale 4 do opisu łuku, można podejść na wiele sposobów. Do obliczeń zastosowano zapis, w którym znany jest środek łuku oraz jego promień. Punkty tworzące łuk można zapisać w postaci wektorowej, jako wektory łączące środek łuku z wybranym punktem leżącym na łuku. Każdy z takich wektorów, tworzy pewien kąt z dodatnią półosią X. Zatem można zapisać jedynie owe kąty, ponieważ długości wektorów są równe promieniowi łuku. Stosując sortowanie zbioru punktów na podstawie kątów jakie tworzą ich reprezentacje wektorowe z osią X uzyskano analogiczny podział na fragmenty jak w przypadku linii.

Istotna jest tutaj kolejność zapisu początku i końca. Dwa punkty opisowe mogą reprezentować dwa różne łuki, dlatego trzeba zdecydować o kierunku łuku. W prezentowanym modelu obliczeniowym zastosowano kierunek przeciwny do ruchu wskazówek zegara, co jest zgodne z naturalnym pomiarem kąta względem dodatniej półosi X.



Rysunek 6.3. Opis łuku stosowany w aplikacji

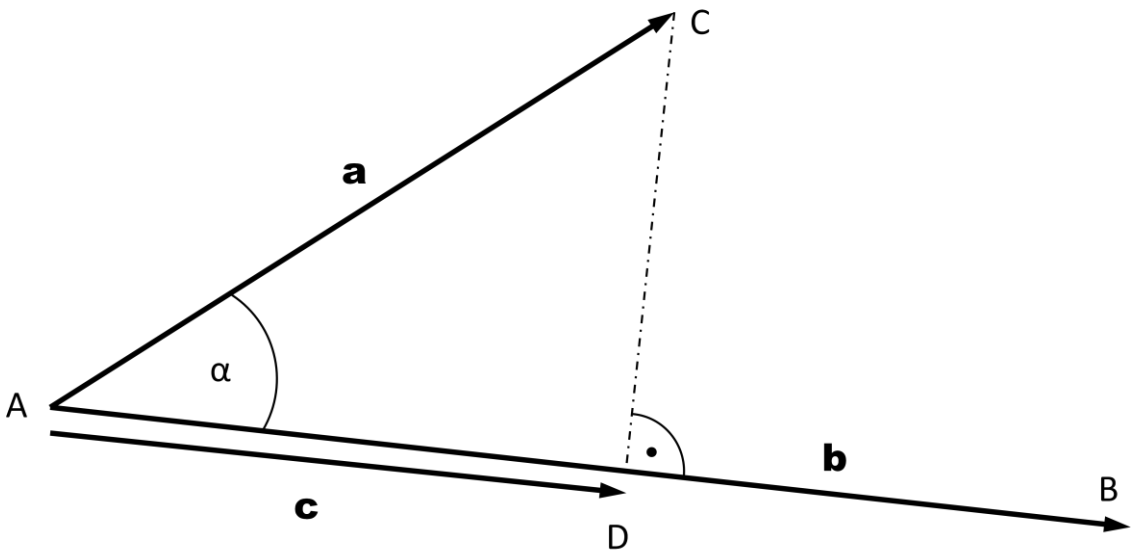
Warto zaznaczyć, że obliczane w tej metodzie kąty są z zakresu od -180 od +180, co podczas sortowania sprawia pewien problem. Przykładowo dla łuku znajdującego się w II i III ćwiartce układu współrzędnych, którego początek w zapisie kątowym wynos 170°, a koniec -170° sortowanie w porządku rosnącym zamieniłoby miejscami początek i koniec. Powstałby w ten sposób łuk po przeciwnej stronie, którego kąt rozwarcia wyniósłby 340°. W związku z tym należało przyjąć, że w zapisie kątowym każdy punkt łuku musi posiadać wartość większą niż punkt początkowy, co wymaga dodania kąta pełnego w prezentowanym przykładzie. Wówczas opisywany łuk rozpoczyna się w kącie 170°, a kończy w 190°. W obliczeniach kąty te zapisane są oczywiście w radianach, stopnie zostały tu zastosowane dla lepszego zobrazowania.

6.3. Algorytmy obliczeniowe

Opisane wyżej funkcjonalności prymitywów wymagają zastosowania wielu ciekawych algorytmów geometrii analitycznej. Obliczenia wektorowe, obliczenia przecięć, sprawdzanie przynależności punktu do prymitywu to tylko niektóre z wymaganych zagadnień.

6.3.1 Rzut punktu na prostą przechodzącą przez dwa punkty

Jest to jeden z najczęściej używanych algorytmów i stanowi podstawę do budowy kolejnych, takich jak np. obliczanie przecięcia łuku i prostej. Podstawą teoretyczną tego algorytmu jest iloczyn skalarny wektorów. Wektory zostały zaznaczone pogrubieniem, a iloczyn wektorowy oznaczony jest znakiem kropki.



Rysunek 6.4. Rzutowanie wektorów

Na powyższym rysunku przedstawiono rzutowanie wektora $\boldsymbol{a}$ na wektor $\boldsymbol{b}$ czego wynikiem jest wektor $\boldsymbol{c}$. Jest to równoznaczne z rzutowaniem prostopadłym punktu C na odcinek |AB|, czego wynikiem jest punkt D. Z trygonometrii można zapisać:

$$\cos(\alpha) = \frac{|\boldsymbol{c}|}{|\boldsymbol{a}|} \tag{2}$$

$$|\boldsymbol{c}| = |\boldsymbol{a}| \cos(\alpha) \tag{3}$$

Mnożąc zależność (3) przez $|\boldsymbol{b}|$ i jednocześnie dzieląc przez $|\boldsymbol{b}|$ otrzymamy zależność na długość wektora $\boldsymbol{c}$ z wykorzystaniem iloczynu skalarnego:

$$|\boldsymbol{c}| = \frac{|\boldsymbol{a}||\boldsymbol{b}|\cos(\alpha)}{|\boldsymbol{b}|} \tag{4}$$

$$|\boldsymbol{c}| = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{|\boldsymbol{b}|} \tag{5}$$

Ostatnim krokiem niezbędnym do wyznaczenia wektora $\boldsymbol{c}$ jest pomnożenie wektora $\boldsymbol{b}$ przez skalar, będący współczynnikiem skali pomiędzy wektorami $\boldsymbol{b}$ i $\boldsymbol{c}$. W wyniku tej operacji uzyskamy wektor $\boldsymbol{c}$ będący przeskalowanym wektorem $\boldsymbol{b}$. Oznaczymy współczynnik jako u:

$$u = \frac{|\boldsymbol{c}|}{|\boldsymbol{b}|} \tag{6}$$

$$\boldsymbol{c} = u\boldsymbol{b} \tag{7}$$

Podstawiając do (7), zależności (6) i (5) uzyskamy końcowy wzór na obliczenie wektora $\boldsymbol{c}$:

$$\boldsymbol{c} = \frac{|\boldsymbol{c}|}{|\boldsymbol{b}|}\boldsymbol{b} \quad \rightarrow \quad \boldsymbol{c} = \frac{\frac{\boldsymbol{a} \cdot \boldsymbol{b}}{|\boldsymbol{b}|}}{|\boldsymbol{b}|}\boldsymbol{b} \quad \rightarrow \quad \boldsymbol{c} = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{|\boldsymbol{b}|^2}\boldsymbol{b} \tag{8}$$

Co można dalej zapisać jako:

$$\boldsymbol{c} = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\boldsymbol{b} \cdot \boldsymbol{b}}\boldsymbol{b} \tag{9}$$

Iloczyn skalarny wektora z samym sobą jest bowiem kwadratem jego długości. Powyższa zależność jest bardzo prosta obliczeniowo z uwagi na występowanie jedynie prostych operacji matematycznych takich jak dodawanie, mnożenie i dzielenie. Nie wymaga stosowanie funkcji trygonometrycznych czy pierwiastkowania. Jest więc szybka obliczeniowo dla procesora. Prostota obliczeń wynika z własności iloczynu skalarnego, który analitycznie w przestrzeni $\mathbf{R}^2$ dany jest wzorem:

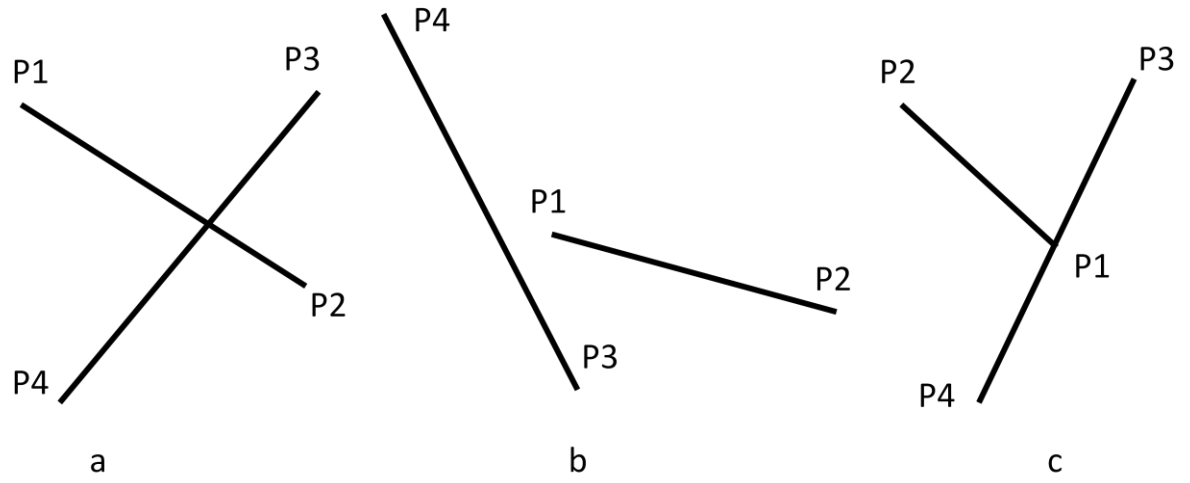
$$\boldsymbol{a} \cdot \boldsymbol{b} = a_x b_x + a_y b_y \tag{10}$$

6.3.2 Przecinanie się odcinków

Wyznaczanie konturu wymaga wielokrotnego liczenia przecięć odcinków. Zasada działania prezentowanego algorytmu jest prosta z uwagi na podejście geometryczne. Nie jest to jednak z pewnością najlepsze rozwiązanie jeśli chodzi o wydajność i złożoność obliczeniową. Wykorzystuje się tutaj równanie prostej w postaci:

$$Ax + By + C = 0 \tag{11}$$

Równanie prostej w postaci funkcji liniowej stanowi problem w przypadku linii równoległej do osi Y. W przypadku postaci (11) nie ma takiego problemu. Algorytm zakłada wyznaczenie równań obu prostych, następnie rozwiązanie układu równań metodą wyznacznikową oraz sprawdzenia przynależności obliczonego punktu do obu odcinków, których dotyczyły obliczenia.



Rysunek 6.5. Możliwości wzajemnego położenie dwóch odcinków

Wyznaczenie zależności umożliwiającej wyliczenie współczynników równania prostej na podstawie współrzędnych punktów P1 i P2 sprowadza się do obliczenia układu równań linowych. W przypadku, gdy różnica odciętych, punktów P1 i P2 wynosi 0, odcinek jest równoległy do osi Y i można od razu podać współczynniki:

$$\begin{cases} A = 1 \\ B = 0 \\ C = -P1_x \ \ lub \ \ C = -P2_x \end{cases} \tag{12}$$

W pozostałych przypadkach prosta przechodząca prze punkty P1 i P2 może być opisana funkcją liniową. Wówczas współczynnik *B* przyjmuje wartość 1.

$$\begin{cases} AP1_x + P1_y + C = 0 \\ AP2_x + P2_y + C = 0 \end{cases} \tag{13}$$

Wyznaczając z pierwszego równania *C* następnie odejmując równania stronami i dokonując dalszych przekształceń można zapisać:

$$\begin{cases} A = -\dfrac{P1_y - P2_y}{P1_x - P2_x} \\ C = -AP1_x - P1_y \end{cases} \tag{14}$$

Należy zastosować zależność (12) lub (14) dla obu odcinków i wyznaczyć współczynniki prostych przechodzących przez pary punktów P1 i P2 oraz P3 i P4. Następnie wyliczyć punkt przecięcia za pomocą układu równań w postaci:

$$\begin{cases} A_1 x + B_1 y = -C_1 \\ A_2 x + B_2 y = -C_2 \end{cases} \tag{15}$$

Gdzie:

A_1, B_1, C_1 – współczynniki pierwszej prostej

A_2, B_2, C_2 – współczynniki drugiej prostej

x, y – współrzędne szukanego punktu przecięcia

Układ równań (15) można łatwo wyliczyć metodą wyznaczników wyznaczając:

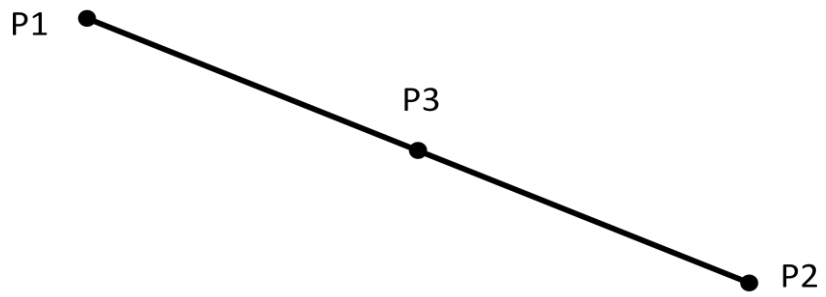
$$\begin{cases} W = A_1 B_2 - B_1 A_2 \\ W_x = B_1 C_2 - C_1 B_2 \\ W_y = C_1 A_2 - A_1 C_2 \end{cases} \tag{16}$$

Wówczas punkt przecięcia można wyliczyć z następujących zależności:

$$\begin{cases} x = \dfrac{W_x}{W} \\ y = \dfrac{W_y}{W} \end{cases} \tag{17}$$

Jak widać wyliczenie punktu przecięcia jest proste, wymaga jedynie obliczenia współczynników z równań (12) lub (14) dla obu odcinków, następnie obliczenia punktu z zależności (16) i (17).

Sprawdzenie przynależności punku do obu odcinków opiera się na założeniu, że punkt leży na prostych wyznaczonych przez punkty P1 i P2 oraz P3 i P4, co wynika z przebiegu obliczeń.



Rysunek 6.6. Położenie sprawdzanego punktu P3 na odcinku |P1 P2|

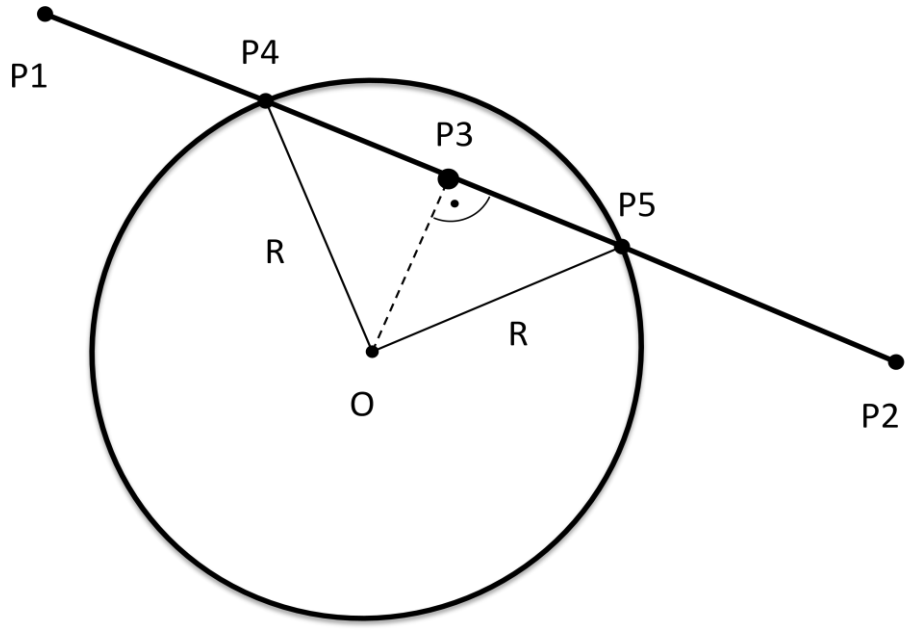
Sprawdzenie przynależności punktu P3 do odcinka |P1 P2| wymaga obliczenia wartości logicznej następującego wyrażenia:

$$\min(P1_x, P2_x) \geq P3_x \geq \max(P1_x, P2_x) \ \wedge \ \min\left(P1_y, P2_y\right) \geq P3_y \geq \max\left(P1_y, P2_y\right) \quad (18)$$

Postępowanie jest oczywiście poprawne tylko w przypadku współliniowości punktów P1, P2 oraz P3. W innym przypadku należałoby najpierw sprawdzić czy punkty są współliniowe.

6.3.3 Przecinanie linii i łuku

Obliczając przecinanie linii i łuku należy rozważać prostą i okrąg. Po uzyskaniu wyniku należy sprawdzić przynależność punktów przecięć zarówno do odcinka jak i do łuku.



Rysunek 6.7. Przecinanie okręgu o ośrodku w punkcie O i prostej przechodzącej przez punkty P1 i P2

Stosując przedstawiony wcześniej algorytm rzutowania prostokątnego można łatwo rozwiązać pokazany na rysunku 6.7. problem. W pierwszym kroku, należy zrzutować punkt centralny okręgu O na prostą przechodzącą przez punkty P1 i P2. Powstały w ten sposób punkt P3 jest jednakowo odległy od punktów P4 i P5. Można zatem korzystając z twierdzenia pitagorasa obliczyć długość odcinka |P3 P4|, następnie opisanym wcześniej skalowaniem wektorów wyznaczyć wektor o długości równej długości odcinka |P3 P4| i kierunku zgodnym z kierunkiem |P1 P2|. Znając punkt P3 można teraz przy użyciu wspomnianego wektora wyliczyć punkt P4, a odwracając ten wektor wyznaczyć również P5.

Obliczanie punków przecięć nie ma oczywiście sensu w przypadku, gdy odległość punktu P3 od środka okręgu O jest większa niż promień R tego okręgu. W przypadku, gdy jest równa istnieje tylko jeden punkt wspólny, który jest w tym przypadku punktem styczności, ale nie jest błędem obliczanie dwóch identycznych punktów za pomocą zaprezentowanego algorytmu. Biorąc pod uwagę omówiony wcześniej problem obliczeń zmiennoprzecinkowych, istnieje bardzo małe prawdopodobieństwo, że punkty P4 i P5 będą identyczne.

Sprawdzenie przynależności punków P4 i P5 do odcinka |P1 P2| jest tutaj analogiczne jak poprzednio. Wnioskowanie o przynależności tych punktów do łuku odbywa się następująco. Obliczone punkty przecięć są przeliczane na reprezentację wektorową względem środka łuku. Następnie obliczane są kąty jakie tworzą te wektory z dodatnią półosią X. Jeśli obliczone kąty znajdują się w przedziale od początkowego do końcowego kąta łuku to można uznać te punkty za rozwiązania tego przypadku. Oczywiście każdy punkt liczony jest oddzielnie, istnieje możliwość, że tylko jeden z obliczonych punktów spełnia wszystkie wymagane kryteria.

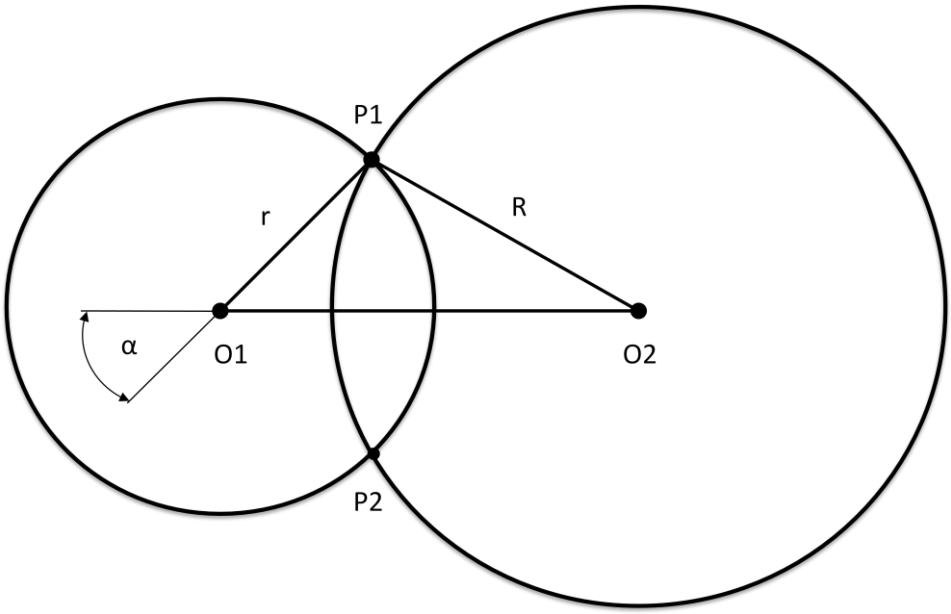
6.3.4 Przecinanie się łuków

Rozważając przecinanie się łuków należy założyć dwa okręgi o środkach w danych punktach O1 i O2 oraz promieniach r i R. Przed rozpoczęciem obliczeń należy wyeliminować sytuację, że okręgi są rozłączne zewnętrznie lub wewnętrznie bądź styczne. Okręgi muszą spełniać następujące warunki:

$$|\overline{O1\ O2}| > |R - r| \quad \wedge \quad |\overline{O1\ O2}| < R + r \tag{19}$$

Pierwszy warunek, gwarantuje że okręgi nie są wewnętrzne natomiast drugi sprawdza czy nie są rozłączne zewnętrznie. Nierówności ostre eliminują możliwość styczności zarówno wewnętrznej jak i zewnętrznej.

Na poniższym rysunku przedstawiono przypadek przecinania się okręgów, którego dotyczą przytoczone tu rozważania. Algorytm zakłada obliczenie kąta α z twierdzenia kosinusów, następnie konstrukcję wektora o długości *r* i kierunku odcinka |O1 O2|. W ostatnim kroku należy obrócić ten wektor o ±*α* i za pomocą dwóch powstałych w ten sposób wektorów, wyznaczyć punktu P1 i P2.



Rysunek 6.8. Wzajemne przecinanie się okręgów

Dla uproszczenia zapisu odległość między środkami okręgów można oznaczyć jako *d*. Wówczas twierdzenie cosinusów dla przypadku z rysunku 6.8. przyjmie postać:

$$R^2 = d^2 + r^2 - 2dr\cos(\alpha) \tag{20}$$

Po przekształceniu zależności (20) można otrzymać:

$$\alpha = arccos\left(\frac{d^2 + r^2 - R^2}{2dr}\right) \tag{21}$$

Wykorzystując skalowanie wektorów przedstawione w rozdziale 6.3.1. można wyznaczyć wektor o długości *r* i kierunku odcinka |O1 O2|, zaczepiony w punkcie O1. Jako typ wektorowy zastosowano liczby zespolone, których interpretacja geometryczna jest w tym przypadku bardzo pomocna. Moduł liczby zespolonej to długość wektora jaki reprezentuje,

a argument to kąt jaki tworzy ten wektor z dodatnią półosią rzeczywistą. W tym przypadku oś rzeczywista traktowana jest jako geometryczna oś X, natomiast oś urojona jako Y. Można zatem obliczyć argument wyznaczonej poprzednio liczby, następnie dodać do niego kąt α i korzystając z postaci wykładniczej liczby zespolonej, skonstruować liczbę o tym samym module i zmienionym argumencie. Wystarczy jeszcze zastosować tę samą operację odejmując kąt α, aby uzyskać dwa wektory zaczepione w punkcie O1 i wskazujące na punkty P1 i P2. Opisana operacja jest łatwa w implementacji, gdy dysponuje się biblioteką umożliwiającą obliczenia na liczbach zespolonych. Poniższy fragment programu prezentuje funkcję z tej biblioteki.

```
… polar( r, ArgO1_O2 + A );
… polar( r, ArgO1_O2 - A );
```

Funkcja *polar()* zwraca liczbę zespoloną o podanym module i argumencie. *ArgO1_O2* jest argumentem liczby stanowiącej interpretacje wektora o początku w punkcie O1 i końcu w punkcie O2. *A* jest tutaj kątem α widocznym na rysunku 6.8. Zastosowanie takiego podejścia znacznie poprawia czytelność kodu programu.

Ostatnim etapem wyznaczania punktów wspólnych jest oczywiście weryfikacja ich przynależności do obu przecinanych łuków. Odbywa się to analogicznie jak opisano poprzednio. Reprezentacja wektorowa punktów przecięć względem środków łuków jako liczb zespolonych ułatwia wyznaczanie kątów jakie tworzą te wektory z dodatnią półosią X. Funkcja *arg()* zwraca argument liczby zespolonej. Biblioteką obsługującą obliczenia zespolone jest standardowa biblioteka C++ o nazwie *complex*.

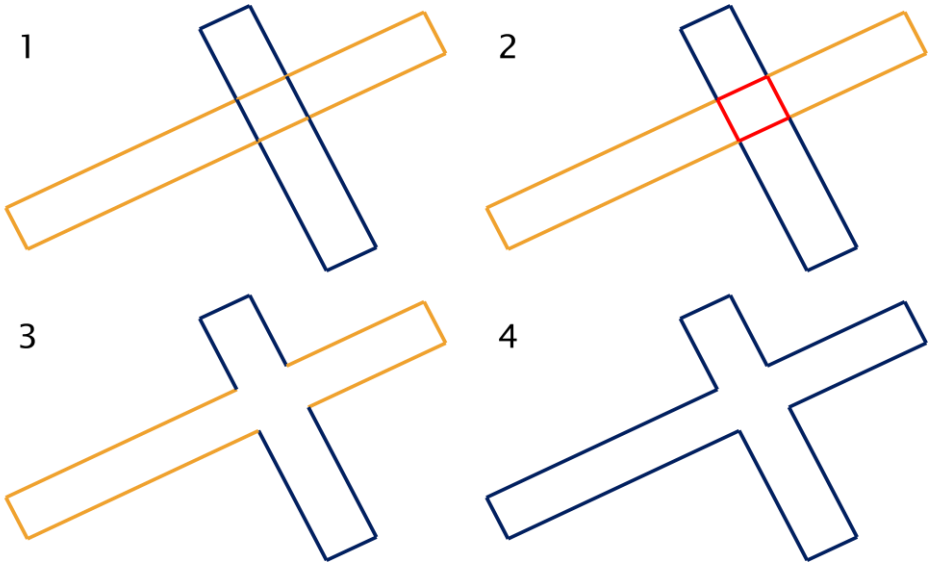
6.3.5. Wyznaczanie wspólnego konturu

Wszystkie opisane do tej pory struktury danych oraz algorytmy obliczeniowe zostały wprowadzone w celu prowadzenia obliczeń wspólnego konturu elementów modelu. Istotę problemu dobrze obrazuje przykład widoczny na rysunku 4.14. w rozdziale poświeconym możliwościom oprogramowania.

Obliczenia prowadzone są wyłącznie na obiektach klasy *PCBObject*, czyli posiadających pole powierzchni ograniczone pewnym konturem. Każdy element modelu należący do wspomnianej klasy, opisany jest pewnym zbiorem prymitywów tworzących jego

kontur. Obiekty te posiadają również metody umożliwiające obliczenie i zwrócenie takiego zbioru.

Kolejną ważną funkcjonalnością tych obiektów jest możliwość sprawdzenia, czy pewien dany punkt znajduje się w ich wnętrzu. Na etapie wspomnianych obliczeń wprowadzona zostaje kompensacja promienia narzędzia, jako pewne odsadzenie od nominalnego konturu. Każdy element posiada charakterystyczne wymiary, na podstawie których wyznacza zbiór prymitywów, może go zatem obliczyć dla wymiarów powiększonych o promień narzędzia. Również sprawdzanie punktu odbywa się z uwzględnieniem odsadzenia.



Rysunek 6.9. Kolejne etapy obliczania wspólnego konturu

Powyższy rysunek przedstawia kolejne etapy algorytmu. Na początku istnieją dwa różne kontury, które na siebie zachodzą. Następuje obliczenie przecięć prymitywów opisujących te kontury. Powstałe punkty przecięć są sortowane w kierunku od początku do końca danego prymitywu. Powstałe w ten sposób pary punktów dzielą prymitywy na fragmenty. Każdy z tych fragmentów znajduje się w całości wewnątrz lub na wewnątrz innych konturów. Można to zobrazować rysując linię na mapie politycznej. Granice państw podzielą ją na fragmenty, z których każdy znajdzie sie w całości w którymś z państw. Nie będzie takiego fragmentu, który jednocześnie znajdzie się w dwóch państwach. Można zatem sprawdzić czy środkowy punkt fragmentu prymitywu należy do danego konturu. Jeśli należy, oznacza to, że powinien zostać usunięty, ponieważ przecina kontur. W przeciwnym wypadku fragment jest prawidłowy i powinien zostać uwzględniony w wyniku. Na powyższym rysunku na czerwono zaznaczono fragmenty, które muszą zostać usunięte.

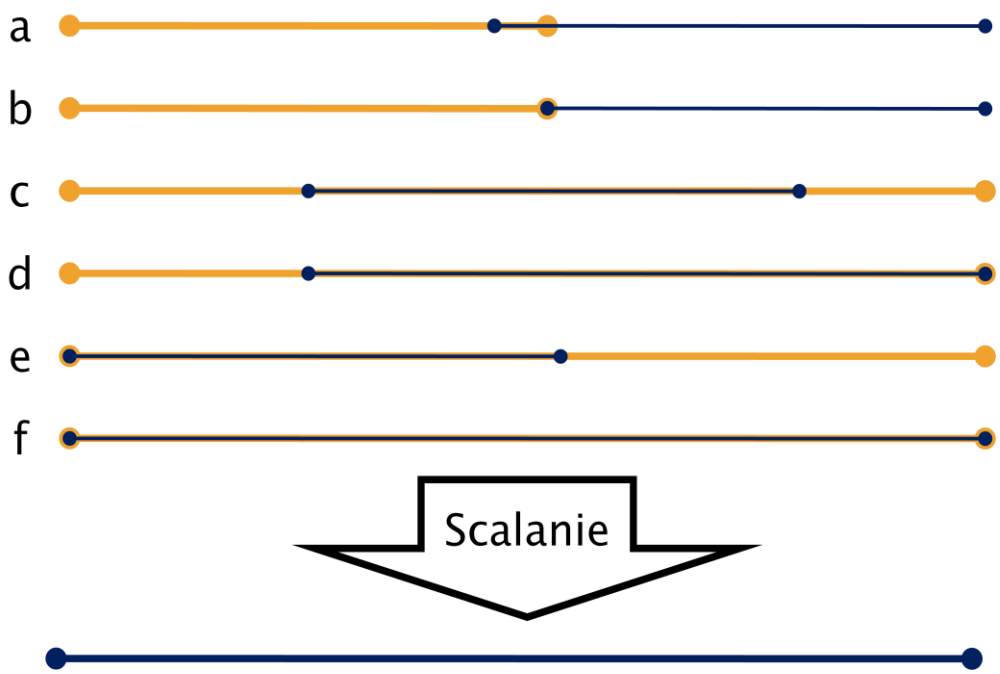
Po zastosowaniu powyższej metody dla wszystkich prymitywów tworzących model, powstaje pewien zbiór fragmentów, co obrazuje rysunek 6.9. na trzeciej pozycji. Aby można było wykonać grawerowanie potrzebne jest jeszcze ułożenie tych fragmentów w odpowiedniej kolejności, tak aby koniec poprzedniego pokrywał się z początkiem następnego. Fragmenty spełniające ten warunek są traktowane teraz jako jeden kontur, co jest pokazana na rysunku 6.9. na ostatniej pozycji. Jeśli odległość do kolejnego fragmentu jest zbyt duża, oznacza to, że należy rozpocząć wykonywanie kolejnego konturu. Generator ścieżki narzędzia wyszukuje w takim przypadku pierwszy najbliższy niewykonany fragment i od niego rozpoczyna tworzenie kolejnego konturu. Bardzo ważną rolę odgrywają tu zaprezentowane wcześniej makra realizujące operacje porównania. Z uwagi na błędy zaokrągleń ciężko jednoznacznie stwierdzić, że dwa fragmenty znajdują dokładnie jeden po drugim. Z punku widzenia obróbki, jeśli odległość między końcem jednego prymitywu, a początkiem następnego jest mniejsza lub równa promieniowi narzędzia, to nie ma sensu wykonywać wyjazdu z materiału i najazdu nad kolejny kontur, ponieważ osiągnięty zostanie dokładnie taki sam efekt, jak w przypadku, kiedy narzędzie pojedzie do tego punktu ruchem roboczym.

6.3.6. Scalania prymitywów

Scalanie prymitywów w końcowym etapie wyznaczania ścieżki narzędzia jest bardzo istotne. Mowa tu o sytuacji, w której dwie linie lub dwa łuki posiadają część wspólną. Nie można wówczas liczyć ich punktów przecięć, ponieważ jest ich nieskończenie wiele. Ponadto istnieje problem zbudowania z nich ścieżki narzędzia, gdyż nie można ich wykonać jeden po drugim. Narzędzie wykonałoby jeden ruch roboczy, następnie musiałoby się cofnąć i wykonać ruch po tej samej ścieżce. W szczególnym przypadku, takie współliniowe prymitywy tworzą łańcuch. Również tutaj narzędzie będzie wykonywać niepożądane ruchy. W przypadku obróbki prowadzonej na robocie przemysłowym, dla uzyskania większej dokładności, następuje zatrzymanie po osiągnięciu zadanego punktu. Zatem wykonanie prostego odcinka, w postaci serii fragmentów z wielokrotnym zatrzymaniem się narzędzia jest również pewnego rodzaju błędem. Generuje to również zbędny kod programu obróbki.

Prymitywy posiadają metody umożliwiające ich scalanie. Najpierw sprawdzana jest ich zgodność, tzn. czy oba są liniami lub łukami. Następnie współliniowość, która w przypadku łuku sprowadza się do wspólnego środka i takiej samej wartości promienia.

W kolejnych etapach sprawdzane są kolejne warunki sprawdzające wzajemne położenie prymitywów. Należy sprawdzić czy punkty końcowe jednego znajdują się we wnętrzu drugiego oraz jaka jest ich liczba i rozmieszczenie. Obrazuje to poniższy rysunek. Zaznaczono na nim możliwe przypadki w których konieczne jest przeprowadzenie scalania.



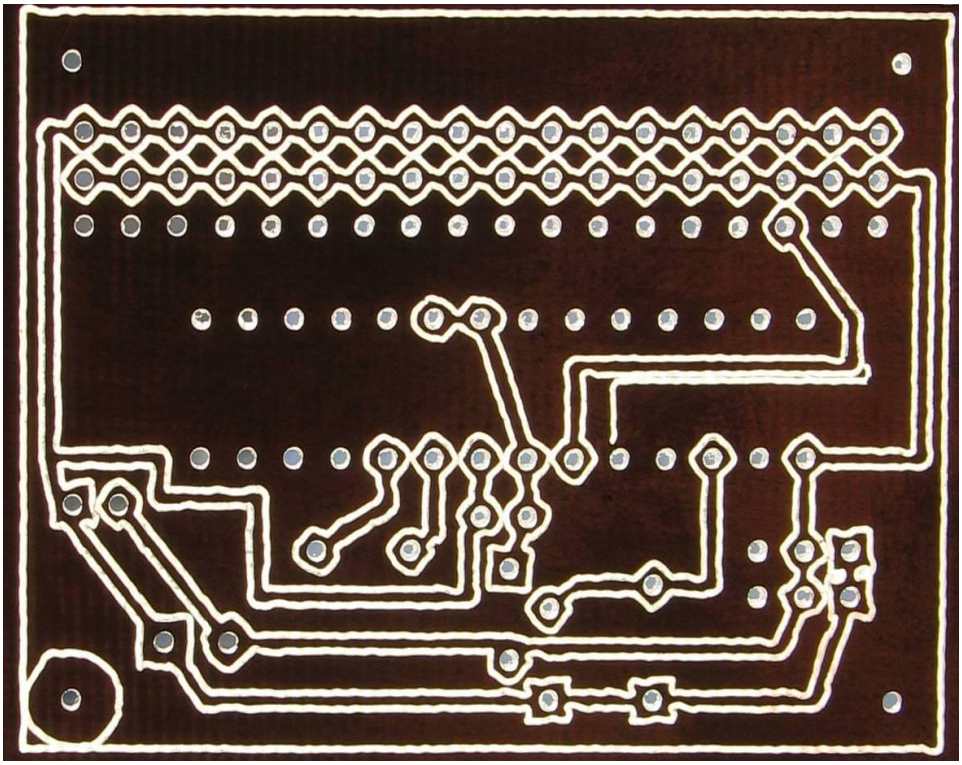
Rysunek 6.10. Możliwe przypadki scalania

Powyższy rysunek przedstawia Kilka możliwych przypadków scalania prymitywów liniowych. Podobne przypadki można by przedstawić dla łuków. Metoda *Join*, realizująca tą operację wykonywana jest na rzecz jednego prymitywu i jako argument przyjmuje wskaźnik na inny. Jeśli spełnione są opisane wcześniej warunki zostaje obliczony prymityw stanowiący sumę obu wspomnianych prymitywów. Metoda *Join* zmienia w tym przypadku kształt prymitywu na rzecz którego została wywołana oraz zwraca wartość logiczną informującą, że dokonano scalenia. Przeprowadzenie tej operacji dla całego zbioru prymitywów wymaga *zewnętrznej obsługi*. Generator ścieżki narzędzia wykonuje tę funkcję dla kolejnych par prymitywów i w przypadku zwrotu *true* usuwa prymityw który został wysłany jako argument funkcji *Join*. Scalony prymityw pozostaje w zbiorze i może podlegać kolejnym scaleniom.

7. WYNIKI TESTÓW MASZYNOWYCH

7.1. Robot KUKA

Na robocie KUKA przeprowadzono pełne testy wykonywania płyty PCB. Wiercenie za pomocą wiertła 1.0mm oraz grawerowanie frezem grawerskim o kącie wierzchołkowym 15°. Efekt końcowy zaprezentowano poniżej.

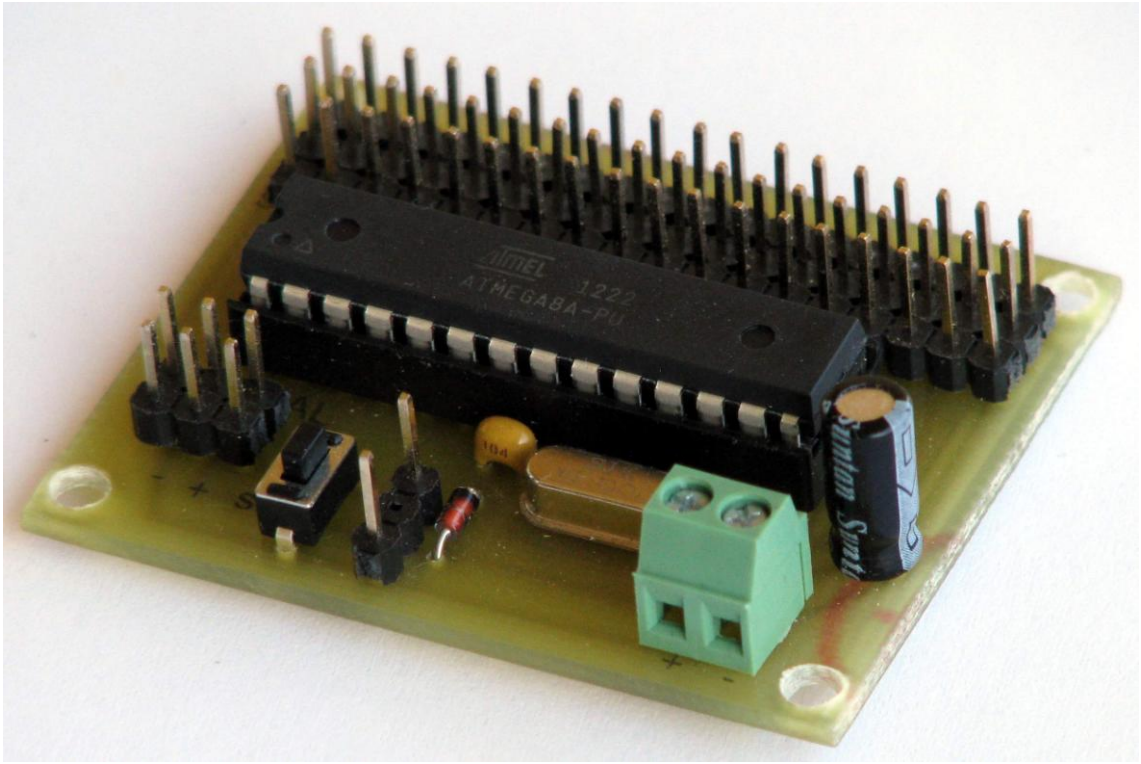


Rysunek 7.1. Prototyp PCB wykonany na robocie KUKA

Na przedstawionym powyżej rysunku widać liczne zniekształcenia. Linie proste są poszarpane, a małe okręgi przypominają zaokrąglone kwadraty. Zniekształcenia na płaszczyźnie roboczej są widoczne gołym okiem. Podobne nieliniowości ruchu robota występują również w osi Z. Skutkuje to zróżnicowaniem głębokości frezowania, a wręcz skokowymi zmianami obciążenia ostrza. Takie warunki pracy doprowadziły do wykruszenia się ostrza narzędzia, czego wynikiem jest niedokończony kontur, widoczny na rysunku.

Wiercenie obyło się bez strat. Wiertło o średnicy 1.0mm zniosło nieliniowe prowadzenie, a otwory zostały wykonane z dostateczną dokładnością umożliwiającą bezproblemowy montaż elementów elektronicznych. Najtrudniejszym fragmentem obwodu drukowanego przedstawionego na rysunku 7.1. jest siatka otworów o wymiarach 18x3. Rozstawy między otworami wynoszą tutaj dokładnie 0.1 cala co jest standardowym odstępem

pól lutowniczych w technologii przewlekanej PCB. Gdyby otwory w tej siatce były wykonane z nieznacznie innym rozstawem, wówczas trudno byłoby osadzić tam listwę *golgpin*. Również współosiowość otworów ma znaczenie. Podczas ręcznego wiercenia przy użyciu mini wiertarki, otwory nie są prostopadłe do płytki PCB, co również utrudnia montaż wspomnianej listwy. Poniżej przedstawiono gotowy zmontowany układ.

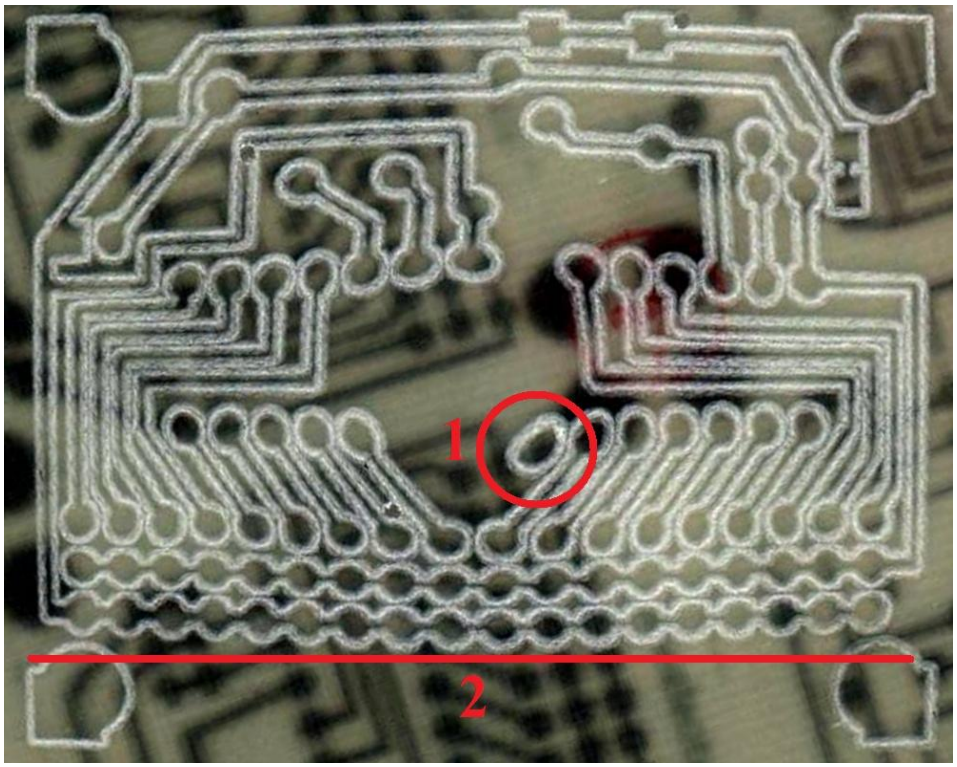


Rysunek 7.2. Serwokontroler *Servo18* – projekt własny

Przedstawiony powyżej układ jest programowanym sterownikiem 18 serwomechanizmów modelarskich. Komunikuje się z komputerem lub innymi urządzeniami za pomocą interfejsu szeregowego. Możliwe jest adresowanie i łączenie szeregowo wielu takich urządzeń dla zwiększenia ilości obsługiwanych serwomechanizmów. Próby maszynowe prowadzone były dla tego właśnie projektu i został on tutaj pokazany w celach poglądowych. Więcej na temat projektu można przeczytać na stronie domowej Autora: *http://radtech.cba.pl*.

7.1. Tokarka z układem sterowania Sinumerik

Na tokarce przeprowadzono jedynie próby grawerowania. Wiercenie nie było możliwe z uwagi na brak odpowiedniego mocowania. Testy miały jedynie potwierdzić przewagę maszyny CNC nad robotem przemysłowym. W początkowym teście laminat zamocowany był centralnie na tarczy zamontowanej we wrzecionie maszyny. Efekt pracy tokarki prezentuje poniższy rysunek.

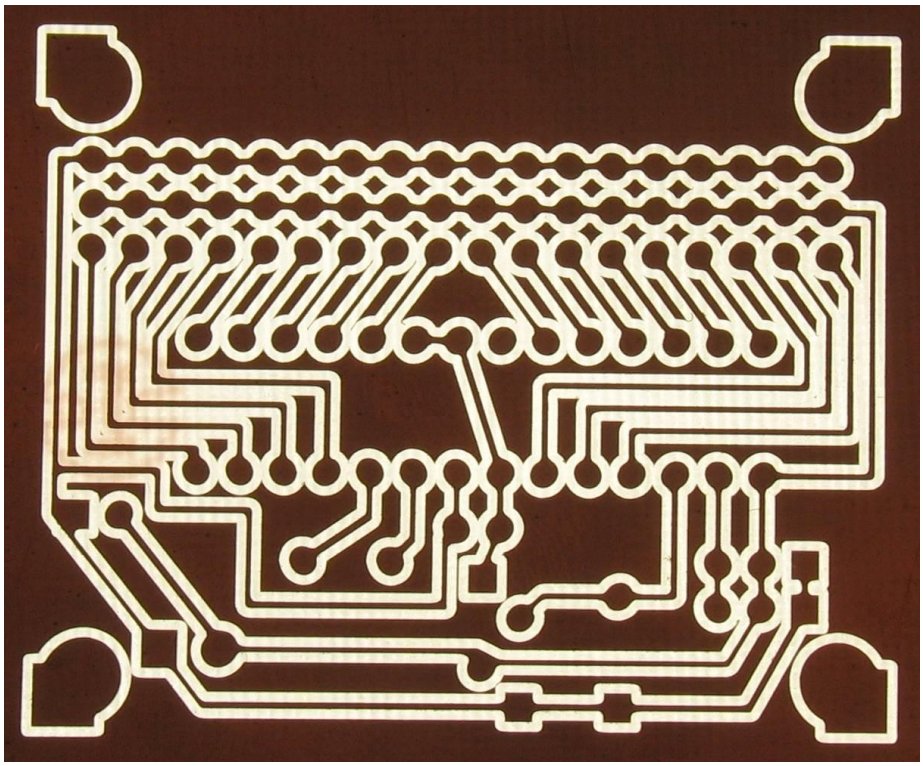


Rysunek 7.3. Rys ścieżek wykonany na tokarce przy mocowaniu centralnym

Na przedstawionym powyżej rysunku widać wyraźne zniekształcenia. Numerem 1 oznaczono pole lutownicze, którego kształt wyraźnie odbiega od okręgu. Numer 2 obrazuje nieliniowość wykonania pól lutowniczych. Zniekształcenia mają związek z mocowaniem laminatu centralnie na tarczy zamocowanej we wrzecionie tokarki. Układ sterowania z trudem wykonuje przejazdy liniowe blisko osi wrzeciona. Niemożliwe jest wręcz wykonanie przejazdów liniowych przechodzących przez oś Z. Z tego powodu konieczne było usunięcie z programu jednego konturu. Linia należąca do niego przechodziła w odległości 0.2mm od środka płytki i tym samym osi wrzeciona.

Laminat na rysunku 7.3. był stary, pozbawiony warstwy miedzi, a na odwrocie posiadał ścieżki naniesione metodą termo transferu. Stąd fatalna jakość tego rysunku.

Kolejny test został przeprowadzony na większej tarczy, gdzie płyta laminatu mogła zostać odsunięta od osi obrotowej maszyny. Punkt centralny płyty został przesunięty o 80mm od osi wrzeciona. Zmniejszono również posuw do 150mm/min. W efekcie otrzymano wynik przedstawiony poniżej.



Rysunek 7.4. Prototyp PCB wykonany na tokarce

Prezentowany powyżej układ został wygrawerowany uszkodzonym wcześniej frezem. Testy na robocie KUKA wykruszyły ostrze frezu, co spowodowało, że jego płaskie zakończenie zwiększyło swoją średnicę. Mimo dużo szerszych rowków pozostawionych przez narzędzie, układ wygląda bardzo dobrze.

Na rysunku 7.4. widać, że w dalszym ciągu widoczne są zniekształcenia. Pola lutownicze nie są idealnie okrągłe, a pomiar wymiaru całkowitego (na rysunku w pionie) wykazał około 1mm różnicy. Nastąpiło powiększenie wymiaru dla obszaru znajdującego się dalej od osi wrzeciona.

7.3. Porównanie robota i tokarki - wnioski

Robot przemysłowy mimo swojej dokładności i powtarzalności nie nadaje się do tego typu obróbki. Grawerowanie PCB jest obróbką wymagającą szczególnej dokładności. Rowek powstały w tej operacji ma około 0.1-0.2 mm, więc zniekształcenia liniowe na tym poziomie są niedopuszczalne. Robot dla osiągnięcia pozycji wraz z utrzymaniem orientacji narzędzia prostopadłej do powierzchni płyty musi zaangażować wszystkie sześć osi obrotowych. Trudno w takim przypadku uzyskać przejazd liniowy. Robot osiąga dokładność w odniesieniu do punktu docelowego, zaś sama ścieżka nie jest już tak dokładna. Kolejną wadą jest niska sztywność konstrukcji, a tym samym podatność na drgania. Szkodzi to narzędziom mocowanym we wrzecionie, co można było zaobserwować podczas prowadzonych testów. Wykruszenie frezu mogło być spowodowane zarówno nieliniowością prowadzenia jak i pojawiającymi się drganiami.

Mimo opisanych wyżej wad, można prowadzić na robocie obróbkę o podobnym charakterze, ale tylko wówczas, kiedy wymagana dokładność będzie na znacznie mniejszy poziomie.

Wiercenie przy użyciu robota KUKA jest jak najbardziej wykonalne o czym świadczą przeprowadzone testy. Można z powodzeniem wykorzystywać robota na tym etapie prototypowania.

Tokarka CNC znacznie lepiej spisuje się podczas grawerowania. Prostopadły układ osi X oraz Z gwarantuje stałą odległość narzędzia od powierzchni płyty. Oczywiście uchwyt montażowy musi umożliwiać precyzyjne mocowanie.

Powstają tutaj jednak zniekształcenia, które również dyskwalifikują tokarkę jako precyzyjne urządzenie do grawerowania płyt PCB. Prawdopodobnie układ sterowania nie radzi sobie z bezwładnością wrzeciona i sterowanie osią C sprawia mu duże problemy. Być może odpowiednie skonfigurowanie układu Sinumerik pozwoliłoby skompensować wspomniane zniekształcenia.

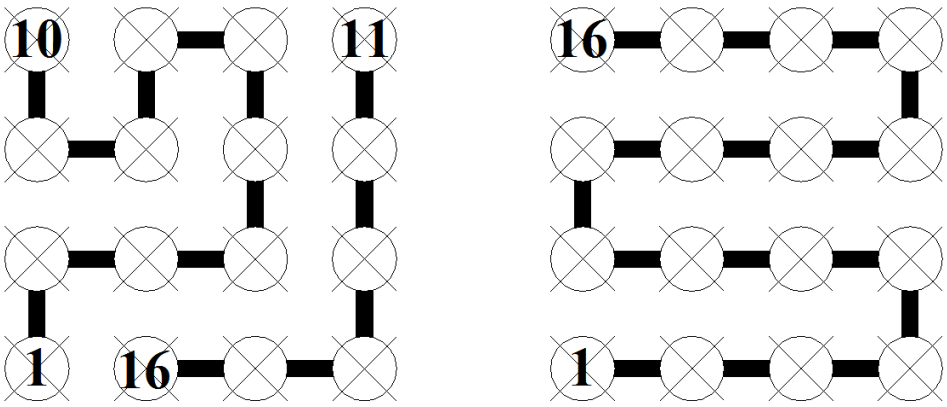
Najlepszym rozwiązaniem zdaje się tu być trzy osiowa obrabiarka, której osie są do siebie prostopadłe. Frezarka lub ploter CNC mają znacznie lepsze uwarunkowania kinematyczne do prowadzenia tego typu obróbki.

8. WNIOSKI I UWAGI KOŃCOWE

8.1. Optymalizacja

Istotnym zagadnieniem podczas generowania ścieżki narzędzia jest optymalizacja. Nie wystarczy wyznaczyć konturu czy listy otworów do wykonania. Należy tak ułożyć kolejne zabiegi aby czas wykonania był jak najkrótszy. Innymi słowy należy zminimalizować udział przejazdów jałowych.

Prezentowane oprogramowanie posiada bardzo prosty system optymalizacji. Wykonywane są przejazdy do najbliższych, nie wykonach jeszcze elementów. W przypadku konturów wybierany jest najbliższy element konturu, od którego rozpoczyna się jego wykonywanie. Po zakończeniu szukany jest kolejny najbliższy element niewykonanego dotąd konturu. W przypadku otworów, również wykonuje się je tak, aby każdy kolejny był jak najbliżej poprzedniego. Dla siatki otworów lub powtarzających się konturów ułożonych w formie siatki o stałych rozstawach, powstaje pewien problem spowodowany błędami zaokrągleń. Istnieje kilka elementów znajdujących się obok, które są jednakowo odległe od poprzednio wykonanego. Błąd zaokrąglenia rozstrzygnąłby w takim przypadku, w którym kierunku biegłaby obróbka. W przypadku siatki otworów spowodowałoby to wiercenie wzdłuż pewnej losowej trajektorii, co powodowałoby powstawanie przerw w siatce, a dalej konieczność powrotu narzędzia. Aby zapobiec takim sytuacją oprogramowanie używa prostego algorytmu rozstrzygającego, który z najbliższych elementów będzie wykonany. Ustawiono tutaj priorytet dla ruchu w osi X. Oznacza to, że jeśli do wyboru będą elementy oddalone o tą samą odległość w osiach X oraz Y, to zostanie wybrany ten przesunięty względem X. Poniższy rysunek obrazuje opisany wyżej problem.



Rysunek 8.1. Losowa oraz optymalna kolejność wykonywania otworów w siatce 4x4

Na rysunku 8.1. przedstawiono dwie możliwe ścieżki przejazdu narzędzia pomiędzy kolejno wykonywanymi otworami. Numerami oznaczono niektóre otwory w siatkach, aby sprecyzować kolejność wykonywania. Pierwsza siatka jest wykonana nieoptymalnie, ponieważ wymagany jest dodatkowy przejazd pomiędzy otworami 10, a 11. Nazywając odległość pomiędzy sąsiednimi otworami *przejazdem jednostkowym* można powiedzieć, że pierwsza siatka wymaga 17 przejazdów jednostkowych dla wykonania wszystkich otworów. Kolejna siatka jest optymalna i wymaga 15 przejazdów jednostkowych, co przy 16 otworach jest minimalną drogą jaką musi pokonać narzędzie w celu wykonania wszystkich otworów. Dokładnie w taki sposób wyznaczana jest ścieżka z uwzględnieniem wspomnianego priorytetu ruchu w osi X. Jak widać, już przy tak małych rozmiarach siatki różnica drogi narzędzia może być zauważalna. Warto jednak zwrócić uwagę na fakt, że istnieje więcej optymalnych rozwiązań tego zadania. Rozwiązania będą się różnić co do umiejscowienia ostatniego otworu siatki. Biorąc pod uwagę wykonywanie kolejnych otworów nie należących do siatki, ma to istotne znaczenie. Należałoby nie tylko optymalnie wykonać siatkę ale też tak ją zakończyć, aby być jak najbliżej kolejnych otworów.

Kontury wykonywane są analogicznie. Jeśli układają się w formie siatki, to priorytet ruchu względem osi X również doskonale się sprawdza. W przypadku konturów punkt rozpoczęcia musi pokrywać się z punktem zakończenia obróbki. Aby zminimalizować przejazdy jałowe należałoby rozważyć odległości pomiędzy punktami początkowo-końcowymi kolejnych konturów. Suma tych odległości powinna być tutaj minimalizowana.

Optymalizacja jest bardzo złożonym i trudnym zagadnieniem. W tej pracy nakreślono jedynie kierunki rozwoju, które powinny zostać uwzględnione na kolejnych etapach budowy aplikacji *PCB CAM Processor*. W pracy nie poruszono szczegółowo tego zagadnienia z uwagi na jego rozległość oraz różnorodność metod optymalizacji. Temat ten powinien zostać podjęty w przyszłości na potrzeby kolejnych prac naukowych. Należy przeanalizować skuteczność różnych technik optymalizacji dla zróżnicowanych modeli oraz sprawdzić na ile obecne metody dalekie są od osiągania rozwiązań optymalnych. Aplikacja stanowi znakomitą bazę do prowadzenia tego typu badań.

8.2. Rozbudowa oprogramowania

Podstawową zaletą tworzenia własnych programów komputerowych jest ich elastyczność, możliwość dostosowania do zmieniających się potrzeb oraz możliwość ciągłej, nieograniczonej rozbudowy.

Prezentowaną aplikację można rozbudowywać w wielu kierunkach. Jednym z kluczowych kierunków rozwoju jest przystosowanie aplikacji do projektowania dwuwarstwowych płyt PCB w technologii SMD. Nie wymaga to dużych zmian w samej aplikacji, natomiast tworzenie takiego rozszerzenia wymaga odpowiedniego przygotowania sprzętowego. Należy dysponować odpowiednimi uchwytami umożliwiającymi precyzyjne przemocowanie płytki dla obróbki drugiej strony. Przeprowadzone do tej pory testy wykazały, że dostępne na uczelni maszyny nie są odpowiednie do wykonywania PCB. Dopiero budowa odpowiedniego stanowiska wyposażonego w ploter lub frezarkę CNC da szansę na dalszy rozwój tego projektu.

Kolejnym etapem rozwoju aplikacji, powinno być wprowadzenie edytora schematów oraz utworzenie bazy obudów elementów elektronicznych. Aplikacja miałaby wówczas funkcjonalności zbliżone do popularnych komercyjnych pakietów takich jak *Cadsoft EAGLE*. Możliwe jest wówczas naszkicowanie schematu, a na jego podstawie stworzenie płyty PCB. Edytor PCB podpowiada, które elementy powinny zostać ze sobą połączone. Wyświetlane są również obudowy tych elementów, co umożliwia odpowiednie rozplanowanie przestrzeni płytki i zapobiega nakładaniu się elementów, co utrudniłoby montaż.

Ostatnim z proponowanych kierunków rozwoju jest umożliwienie importu projektów z innych oprogramowań do projektowania PCB. Przydatny może się również okazać import formatu DXF, umożliwiający wykonywanie detali na podstawie rysunków z programów typu *AutoCAD*.

8.3. Obszary zastosowań

Głównym zastosowaniem prezentowanej aplikacji jest prototypowanie płyt drukowanych. Budowa odpowiedniego stanowiska pozwoliłaby na prowadzenie licznych projektów z zakresu elektroniki. Studenci Akademii mogliby w ramach kół naukowych wykonywać za darmo elektronikę do różnego rodzaju pojazdów czy robotów. Również hobbyści posiadający w domach plotery CNC mogą z powodzeniem zastosować aplikację do tworzenia swoich projektów.

Aplikacja jest uniwersalna. Z powodzeniem można ją wykorzystać do projektowania i wykonywania różnego rodzaju ozdób np. liter ze styroduru (pop. polistyren ekstrudowany). Wymaga to oczywiście odpowiednich narzędzi i stanowiska.

Oprogramowanie PCB CAM Processor ma liczne zastosowania i może służyć jako narzędzie w wielu przyszłych pracach dyplomowych. Może też stać się ich podstawą, wartą wspomnianej wcześniej rozbudowy.

PODSUMOWANIE

Podczas realizacji opisanego w niniejszej pracy projektu napotkano wiele problemów. Głownie były to problemy związane z geometrią obliczeniową. Było również kilka problemów natury czysto programistycznej takich jak: dziedziczenie po klasach abstrakcyjnych, rzutowanie dynamiczne obiektów na typy pochodne czy stosowanie szablonów.

Projekt oprogramowania znacząco się rozrósł podczas prac, osiągając ponad 40 plików źródłowych. Organizacja pracy nad tak dużym projektem jest niemałym wyzwaniem. Bezwątpienia był to największy realizowany dotąd projekt Autora.

Najtrudniejsze było znalezienie sposobu wyznaczania wspólnych konturów. Zajęło to wiele miesięcy. Opracowywano i testowano wiele różnych algorytmów, aż wreszcie został stworzony banalnie prosty algorytm prezentowany w niniejszej pracy.

W efekcie ciężkiej i żmudniej pracy, powstało uniwersalne narzędzie umożliwiające projektowanie oraz prototypowanie PCB. Zaproponowane szablony maszynowe stały się bardzo elastyczną formą opisy składni języka maszyny, co dało możliwość generowania kodu dla niemal dowolnej obrabiarki czy robota przemysłowego.

Efekt końcowy spełnia wszystkie założone cele pracy. Dodatkowo praca zawiera próby z inną niż zakładano platformą, tokarką z układem sterowania Sinumerik. Walidacja interfejsu pokazała jednak, że o ile po stronie oprogramowania czy stosowanej metodologii nie ma uchybień, to po stronie użytych urządzeń występują niedopuszczalne błędy. Poszarpane ścieżki dla robota czy zniekształcenie nieliniowe dla tokarki dyskwalifikują te urządzenia. Autor wyraża jednak gorącą nadzieję, że projekt ten, nie zakończy się na tej pracy i że będzie kontynuowany w przyszłości. Szczególnie istotnym aspektem jest nie samo rozwijanie oprogramowania *PCB CAM Processor*, ale przygotowanie specjalnego, dedykowanego stanowiska do prototypowania PCB. Dałoby to uczelni bardzo dobre zaplecze do tworzenia projektów z dziedziny elektroniki bez korzystania z kosztownych usług firm zewnętrznych. Ponadto aplikacja może mieć również inne zastosowania, o których wspomniano w rozdziale ósmym oraz może być dowolnie rozwijana.

Podsumowując, Autor wyraża zadowolenie przebiegiem prac nad projektem, stwierdza że wiele się nauczył podczas jego realizacji oraz wierzy, że ta praca nie jest końcem rozwoju tego projektu.

BIBLIOGRAFIA

1. Craig J. J.: *Wprowadzenie do robotyki* (tłum. z angielskiego). WNT, Warszawa 1993.
2. Gręboszz Jerzy: *Symfonia C++ - Programowanie w języku C++ orientowane obiektowo*. Oficyna Kallimach, Kraków 1999.
3. Sellers Graham, Richard S. Wright Jr., Nicholas Haemel: *OpenGL. Księga eksperta* (tłum. z angielskiego). Wyd. Helion, Gliwice 2016.
4. Stryczek Roman, Pytlak Bogusław: *Elastyczne programowanie obrabiarek*. Wyd. Naukowe PWN, Warszawa 2011.
5. Szkodny Tadeusz: *Podstawy robotyki*. Wyd. Politechniki Śląskiej, Gliwice 2012.
6. Zdanowicz Ryszard: *Podstawy robotyki*. Wyd. Politechniki Śląskiej, Gliwice 2012.
7. Siwiec Radosław: *Opracowanie postprocesora dla generowania ścieżki narzędzia robota przemysłowego*. Artykuł konferencyjny VI Międzynarodowej Konferencji Studentów oraz Doktorantów „Inżynier XXI Wieku". Wydrukowany w Monografii Konferencji. Wyd. naukowe Akademii Techniczno-Humanistycznej, Bielsko-Biała 2016, s.817-824.
8. Mechanik 4/2012 *MasterCAM RobotMaster*, Zalewski Adam.
9. KUKA Roboter GmbH: *Expert Programming. KUKA System Software (KSS)*, Release 5.2, issued: 26.09.2003.
10. KUKA Roboter GmbH: *Training - Zastosowanie i programowanie robotów przemysłowych*, stan na: 15.12.2015.
11. KUKA Roboter GmbH: *Training – Programowanie robota 1. KUKA System Software 8 – dokumentacja szkoleniowa*, stan na: 27.09.2012.
12. KUKA Roboter GmbH: *KR AGILUS sixx – Instrukcja obsługi*, stan na: 25.03.2015.
13. KUKA Roboter GmbH: *Training – KUKA.Sim 2.2 Pro*, issued: 05.07.2012.
14. KUKA Roboter GmbH: *Training – KUKA.Sim 2.2 Layout*, issued: 05.07.2012.
15. Siemens: *SINUMERIK 840D/840Di/810D - Instrukcja programowania - Podstawy*. Wydanie 03.04.
16. http://www.cplusplus.com/reference/-dokumentacja standardowych bibliotek języka C++.
17. http://www.wxwidgets.org/ - dokumentacja bibliotek wxWidgets.
18. https://www.opengl.org/ - strona domowa projektu OpenGL.
19. http://www.algorytm.org/geometria-obliczeniowa/ - algorytmy geometryczne.
20. http://www.obliczeniowo.com.pl/?id=171 - algorytmy geometryczne.

DODATEK A – KUKA – PROGRAMY POMOCNICZE

Przejazd szybki liniowy – kompletny listing:

```
&ACCESS RVO2
&REL 3
&PARAM EDITMASK = *
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
&PARAM DISKPATH = KRC:\R1\Siwiec Radoslaw
DEF FLin( DestPoint:in )
decl e6pos DestPoint

XP1 = DestPoint

;FOLD LIN P1 Vel=2 m/s CPDAT1 Tool[8] Base[31];%{PE}%R
8.3.31,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, 2:P1, 3:, 5:2,
7:CPDAT1
$BWDSTART=FALSE
LDAT_ACT=LCPDAT1
FDAT_ACT=FP1
BAS(#CP_PARAMS,2)
LIN XP1
;ENDFOLD

END
```

Przejazd szybki liniowy –listing uproszczony, wzbogacony o komentarze:

```
;Deklaracja programu FLin z wejściowym parametrem
DEF FLin( DestPoint:in )
;Deklaracja zmiennej stanowiącej parametr programu
decl e6pos DestPoint
;Podmiana wartości zmiennej XP1 na odebrany parametr
XP1 = DestPoint
;Przejazd liniowy do punktu XP1 z prędkością 2 m/s
LIN P1 Vel=2 m/s CPDAT1 Tool[8] Base[31]
;Zakończenie programu
END
```

Pierwszy przykład prezentuje kod programu wraz ze składnią początkową oraz pełnym zapisem instrukcji ruchu. Taki format pliku wymagany jest, kiedy generowany jest program dla robota. Użytkownik widzi natomiast program w formie uproszczonej. Na eksperckim poziomie dostępu możliwe jest rozwijanie poszczególnych bloków programu. Bloki zapisane są w poniższy sposób:

```
;FOLD
...
;ENDFOLD
```

Przejazd roboczy liniowy:

```
DEF WLin( DestPoint:in )
decl e6pos DestPoint

XP1 = DestPoint

LIN P1 Vel=0.005 m/s CPDAT1 Tool[8] Base[31]

END
```

Przejazd roboczy w interpolacji kołowej:

```
DEF WCirc( AddPoint:in, DestPoint:in )
decl e6pos DestPoint, AddPoint

XP1 = AddPoint      ;Punkt pośredni ruchu
XP2 = DestPoint

CIRC P1 P2 Vel=0.005 m/s CPDAT1 Tool[8] Base[31]

END
```

Programy *WLin* oraz *WCirc* stanowią opis ruchów roboczych z prędkością liniową 0.005 m/s oraz przyspieszeniem 1%. *WCirc* przyjmuje dwa parametry wejściowe: punkt docelowy oraz punkt pośredni. Wszystkie powyższe programy zostały wprowadzone dla uproszczenia zapisu oraz dla umożliwienia edycji parametrów ruchu zwykłemu użytkownikowi. Zmiana parametrów w programie generowanym przez *PCB CAM Processor* wymaga dostępu eksperckiego. Dodatkowo należałoby ustawiać te parametry już w szablonach maszynowych.

Przygotowanie wrzeciona:

```
DEF grindinit( )
;Ustawienia początkowe – blok FOLD – nie podlega edycji
INI
;Przygotowanie robota - powrót do pozycji HOME
PTP HOME Vel=20 % DEFAULT
;Prostopadłe ustawienie wrzeciona
FLin( { A 90,  B 90,  C 90  } )
;Najazd nad płaszczyznę obróbki
FLin( { X 580, Y 400, Z 500 } )

END
```

Powyższy program służy do przygotowania robota do prowadzenia obróbki. Należy ustawić wrzeciono prostopadle do płaszczyzny obróbki oraz najechać nad powierzchnię obrabianą.

Podprogram wiercenia:

```
DEF Drill( StartPoint:in, H:in )
;Deklaracja zmiennej początku wiercenia
decl e6pos StartPoint
;Deklaracja zmiennej głębokości wiercenia
decl int H

XP1 = StartPoint
XP2 = StartPoint

;Obliczenie pozycji końcowej wiercenia
XP1.z = XP1.z –H

;Przejazd nad otwór – dojazd do punktu wiercenia
FLin( StartPoint )
WAIT SEC 0.2 ;Oczekiwanie na ustabilizowanie ramienia

;Przejazd roboczy – wiercenie
LIN P1 Vel=0.005 m/s CPDAT1 Tool[8] Base[31]
wait sec 0.2 ;Oczekiwanie na ustabilizowanie ramienia

;Wyjazd z otworu – powrót na płaszczyznę ruchu
LIN P2 Vel=0.005 m/s CPDAT2 Tool[8] Base[31]

END
```

Powyższy program realizuje sekwencję: najazd nad otwór ruchem szybkim, wgłębienie się w materiał na głębokość H oraz powrót ruchem roboczym na płaszczyznę ruchu. Program ma własne ustawienia ruchu roboczego nie zależne od prezentowanych wcześniej. Dzięki powrotowi na płaszczyznę ruchu można wykonywać program wielokrotnie, jeden po drugim

DODATEK B – KUKA – PRZYKŁADOWY PROGRAM GRAWEROWANIA

```
DEF serwokontroler4_pcb_Mill( )
DECL FRAME Vec
;Ustawienie przesunięcia – kalibracja ukł. współrzędnych
Vec = { X 0, Y 0, Z -0.2 }

;najazd nad płytę, ustawienie wrzeciona
grindinit()

;Najazd nad pierwszy punkt konturu
FLin( transvec( { X -0.115802, Y -0.119124, Z 5 }, Vec ) )
;Zagłębienie w materiał
WLin( transvec( { X -0.115802, Y -0.119124, Z 0 }, Vec ) )

;Przejazdy robocze po konturze
WLin( transvec( { X -0.115012, Y 40.120211, Z 0 }, Vec ) )
WLin( transvec( { X 50.121458, Y 40.119767, Z 0 }, Vec ) )
WLin( transvec( { X 50.110012, Y -0.119295, Z 0 }, Vec ) )
WLin( transvec( { X -0.115802, Y -0.119124, Z 0 }, Vec ) )

;Wyjazd z konturu – powrót na płaszczyznę ruchu
FLin( transvec( { X -0.115802, Y -0.119124, Z 5 }, Vec ) )

;Najazd nad pierwszy punkt konturu
FLin( transvec( { X -0.515806, Y -0.319099, Z 5 }, Vec ) )
;Zagłębienie w materiał
WLin( transvec( { X -0.515806, Y -0.319099, Z 0 }, Vec ) )

;Przejazdy robocze po konturze
WLin ( transvec( { X -0.515008, Y 40.320197, Z 0 }, Vec ) )
WCirc( transvec( { X -0.456434, Y 40.461629, Z 0 }, Vec ),
       transvec( { X -0.315006, Y 40.520213, Z 0 }, Vec ) )
WLin ( transvec( { X 50.321517, Y 40.519765, Z 0 }, Vec ) )
WCirc( transvec( { X 50.462950, Y 40.461173, Z 0 }, Vec ),
       transvec( { X 50.521515, Y 40.319708, Z 0 }, Vec ) )
WLin ( transvec( { X 50.509955, Y -0.319333, Z 0 }, Vec ) )
WCirc( transvec( { X 50.451349, Y -0.460745, Z 0 }, Vec ),
       transvec( { X 50.309934, Y -0.519296, Z 0 }, Vec ) )
WLin ( transvec( { X -0.315807, Y -0.519123, Z 0 }, Vec ) )
WCirc( transvec( { X -0.457222, Y -0.460550, Z 0 }, Vec ),
       transvec( { X -0.515806, Y -0.319119, Z 0 }, Vec ) )

;Wyjazd z konturu – powrót na płaszczyznę ruchu
FLin( transvec( { X -0.515806, Y -0.319119, Z 5 }, Vec ) )

;Dalsze kontury                ...
```

```
;Najazd nad pierwszy punkt konturu
FLin( transvec( { X 5.135785, Y 2.392011, Z 5 }, Vec ) )
;Zagłębienie w materiał
WLin( transvec( { X 5.135785, Y 2.392011, Z 0 }, Vec ) )

;Przejazdy robocze po konturze
WCirc( transvec( { X -0.059999, Y 2.542721, Z 0 }, Vec ),
transvec( { X 5.136089, Y 2.682555, Z 0 }, Vec ) )
WCirc( transvec( { X 5.198474, Y 2.537218, Z 0 }, Vec ),
transvec( { X 5.135785, Y 2.392011, Z 0 }, Vec ) )

;Wyjazd z konturu – powrót na płaszczyznę ruchu
FLin( transvec( { X 5.135785, Y 2.392011, Z 5 }, Vec ) )

;Wyjazd na bezpieczną pozycję końcową
FLin( { X 300, Y 300, Z 600 } )
;Powrót do HOME
FOLD PTP HOME Vel=30 % DEFAULT

END
```

Zaprezentowany program posiada możliwość wprowadzenia przesunięcia o wektor jest to szczególnie przydatne, kiedy potrzebne jest obniżenie narzędzia w celu kalibracji. Można wykonać cały program, następnie widząc że narzędzie nie wykonało poprawnych rowków obniżyć o zadaną wartość kalibracyjną i wykonać program ponownie. Do obsługi przesunięcia kalibracyjnego wykorzystano funkcję *transvec* przyjmującą za argumenty: punkt pierwotny oraz wektor przesunięcia, a zwracającą punkt przesunięty o zadany wektor względem zadanego punktu.

DODATEK C – KUKA – PRZYKŁADOWY PROGRAM WIERCENIA

```
DEF serwokontroler4_pcb_Drill( )
DECL FRAME Vec
DECL INT Deep

;Ustawienie przesunięcia – kalibracja ukł. współrzędnych
Vec = { X 0, Y 0, Z 0 }
;Ustawienie głębokości wiercenia
Deep = 5;

;najazd nad płytę, ustawienie wrzeciona
grindinit()

Drill( transvec( { X 2.5400,  Y 2.5400,  Z 3 }, Vec ), Deep )
Drill( transvec( { X 7.6200,  Y 5.7150,  Z 3 }, Vec ), Deep )
Drill( transvec( { X 11.1125, Y 5.7150,  Z 3 }, Vec ), Deep )
Drill( transvec( { X 15.8750, Y 10.7950, Z 3 }, Vec ), Deep )
Drill( transvec( { X 20.9550, Y 10.7950, Z 3 }, Vec ), Deep )
Drill( transvec( { X 24.7650, Y 12.7000, Z 3 }, Vec ), Deep )
Drill( transvec( { X 27.3050, Y 12.7000, Z 3 }, Vec ), Deep )
Drill( transvec( { X 26.3525, Y 9.8425,  Z 3 }, Vec ), Deep )
Drill( transvec( { X 28.5750, Y 7.6200,  Z 3 }, Vec ), Deep )
Drill( transvec( { X 26.4028, Y 4.7642,  Z 3 }, Vec ), Deep )
Drill( transvec( { X 28.5750, Y 2.5400,  Z 3 }, Vec ), Deep )
Drill( transvec( { X 34.2900, Y 2.5400,  Z 3 }, Vec ), Deep )
Drill( transvec( { X 34.2900, Y 8.8900,  Z 3 }, Vec ), Deep )
Drill( transvec( { X 40.0050, Y 8.2550,  Z 3 }, Vec ), Deep )
Drill( transvec( { X 42.5450, Y 8.2550,  Z 3 }, Vec ), Deep )
Drill( transvec( { X 45.0850, Y 8.2550,  Z 3 }, Vec ), Deep )
Drill( transvec( { X 45.0850, Y 10.7950, Z 3 }, Vec ), Deep )
Drill( transvec( { X 42.5450, Y 10.7950, Z 3 }, Vec ), Deep )
Drill( transvec( { X 40.0050, Y 10.7950, Z 3 }, Vec ), Deep )
Drill( transvec( { X 40.0050, Y 15.8750, Z 3 }, Vec ), Deep )

;Dalsze otwory                    ...

;Wyjazd na bezpieczną pozycję końcową
FLin( { X 300, Y 300, Z 600 } )
;Powrót do HOME
FOLD PTP HOME Vel=30 % DEFAULT

END
```

W zaprezentowanym programie łatwo można zmienić głębokość wiercenia oraz skalibrować położenie narzędzia.

DODATEK D – SINUMERIK - PRZYKŁADOWY PROGRAM GRAWEROWANIA

```
;serwokontroler08_pcb_Mill
;Wybór punktu zerowego, płaszczyzny XY, prog. absolutnego
;posuwu w mm/min oraz wymiarowania na promieniu
N1 G54 G17 G90 G94 DIAMOF
;Załączenie płynnego sterowania przyspieszeniem napędów
;Unikanie udarów, czyli skoków przyspieszenia
N2 SOFT
;Wybór zmierzonego wcześniej narzędzia
N8  T="FREZ" D1
;Uruchomienie wrzeciona dodatkowego na 10000 obr/min
N9  S2=10000 M2=3
Ustawienie posuwu na 500 mm/s
N10 F 500
;Ustawienie wstępne kąta wrzeciona głównego – osi C
N11 SPOS=180
;Uaktywnienie trybu pracy ze sterowną osią C
N12 TRANSMIT
;Przesunięcie układu wsp. o 80mm od osi wrzeciona głównego
N13 TRANS X80

;Najazd nad pierwszy punkt konturu
N14 G1 X-0.469496 Y1.347963 Z2
;Zagłębienie w materiał
N15 G1 Z-0.2

;Przejazdy robocze po konturze
N16 G2 X-1.363743 Y1.992500 I0.109496 J1.094537
N17 G1 X-1.896257 Y1.992500
N18 G2 X-1.896257 Y2.892500 I-1.003743 J0.450000
N19 G1 X-1.363743 Y2.892500
N20 G2 X0.384319 Y1.632568 I1.003743 J-0.450000
N21 G1 X2.289496 Y-4.082963
N22 G2 X2.630000 Y-6.181243 I-0.109496 J-1.094537
N23 G1 X2.630000 Y-7.348757
N24 G2 X2.225034 Y-9.451578 I-0.450000 J-1.003743
...

N31 G1 X1.389405 Y-9.117326
N32 G2 X1.730000 Y-7.348757 I0.790595 J0.764826
N33 G1 X1.730000 Y-6.181243
N34 G2 X1.435681 Y-4.367568 I0.450000 J1.003743
N35 G1 X-0.469496 Y1.347963

;Wyjazd z konturu – powrót na płaszczyznę ruchu
N36 G1 Z2

;Dalsze kontury                ...
```

```
;Najazd nad pierwszy punkt konturu
N553 G1 X-24.355844 Y-15.900000 Z2
;Zagłębienie w materiał
N554 G1 Z-0.2

;Przejazdy robocze po konturze
N555 G2 X-20.900000 Y-19.355844 I2.355844 J-1.100000
N556 G1 X-20.900000 Y-20.100000
N557 G1 X-25.100000 Y-20.100000
N558 G1 X-25.100000 Y-15.900000
N559 G1 X-24.355844 Y-15.900000

;Wyjazd z konturu – powrót na płaszczyznę ruchu
N560 G1 Z2

;Odwołanie przesunięcia ukł. współrzędnych
N561 TRANS X0 Y0
;Zatrzymanie przebiegu wyprzedzającego
N562 STOPRE
;Odwołanie funkcji TRANSMIT
N563 TRAFOOF
;Wyjazd na bezpieczną pozycję końcową
N564 Z100
;Wyłączenie obrotów wrzeciona pomocniczego
N565 M2=5
;Powrót osi C na pozycję zerową
N566 SPOS=0
;Koniec programu, powrót na początek
N567 M30
```

Przedstawiony powyżej program umożliwia pełne wykonanie ścieżek płytki PCB. Jak widać program zawiera ponad pięćset linii kodu, zatem nie sposób zaprezentować tutaj całego programu. W programie tokarka została użyta w trybie pracy ze sterowaną osią C, co umożliwia pracę w trzech osiach współrzędnych XYZ, gdzie płaszczyzna XY znajduje się na czole uchwytu maszyny, a oś Z pokrywa się z osią Z zwykłego układu tokarki. Funkcja TRANS służy do tego, aby wykonywać program na obrzeżu uchwytu tokarki. Daje to lepszą dokładność wykonania. Obróbka prowadzona blisko osi wrzeciona cechuje się dużymi zniekształceniami.

Mimo ustawienia posuwu na poziomie 500 mm/min, program był wykonywany na 30%, czyli rzeczywisty posuw wynosił 150 mm/min (2.5 mm/s). Wszystkie przejazdy szybkie (G0), zostały zastąpione roboczymi (G1), aby zminimalizować prędkości osiągane przez maszynę.

DODATEK E – SINUMERIK - PRZYKŁADOWY PROGRAM WIERCENIA

```
;serwokontroler08_pcb_Drill
;Wybór punktu zerowego, płaszczyzny XY, prog. absolutnego
;posuwu w mm/min oraz wymiarowania na promieniu
N1 G54 G17 G90 G94 DIAMOF
;Załączenie płynnego sterowania przyspieszeniem napędów
;Unikanie udarów, czyli skoków przyspieszenia
N2 SOFT
;Wybór zmierzonego wcześniej narzędzia
N8  T="FREZ" D1
;Uruchomienie wrzeciona dodatkowego na 10000 obr/min
N9  S2=10000 M2=3
Ustawienie posuwu na 500 mm/s
N10 F 500
;Ustawienie wstępne kąta wrzeciona głównego – osi C
N11 SPOS=180
;Uaktywnienie trybu pracy ze sterowną osią C
N12 TRANSMIT
;Przesunięcie układu wsp. o 80mm od osi wrzeciona głównego
N13 TRANS X80

N14 G0 X7.495000 Y4.662500 Z2       ;Najazd nad otwór
N15 G1 Z-3                          ;Wykonanie wiercenia
N16 G1 Z2                           ;Powrót na pł. ruchu

;Dalsze otwory                  ...

;Odwołanie przesunięcia ukł. współrzędnych
N328 TRANS X0 Y0
;Zatrzymanie przebiegu wyprzedzającego
N329 STOPRE
;Odwołanie funkcji TRANSMIT
N330 TRAFOOF
;Wyjazd na bezpieczną pozycję końcową
N331 Z100
;Wyłączenie obrotów wrzeciona pomocniczego
N332 M2=5
;Powrót osi C na pozycję zerową
N333 SPOS=0
;Koniec programu, powrót na początek
N334 M30
```

Program ma analogiczne ustawienia początkowe oraz bloki końcowe, co program grawerowania. Z uwagi na brak odpowiedniego mocowania płyty PCB nie został uruchomiony na tokarce. Prowadzone były jedynie próby na symulatorze, które zakończyły się powodzeniem.