Jakub Nowacki
Sprawozdanie z projektu 'Borsuk malarz'.

1 Motywacja

Jakiś czas temu spotkałem się w internecie z dość ciekawym projektem. Do maszyny CNC, która służyła do wycinania elementów, dorobiono element umożliwiający umocowanie markera. Dzięki temu na wszelkiego rodzaju powierzchniach można było szybko i automatycznie wykonywać bardzo dokładne rysunki.

Stąd zrodził się w mojej głowie pomysł, aby spróbować czegoś podobnego, doczepiając marker do robota borsuk. Już na początku zdawałem sobie sprawę, że raczej nie uda się osiągnąć dużej precyzji, korzystając z takiego rozwiązania, ale wciąż brzmiało to jak ciekawy projekt.

2 Metoda wczytywania rysunku

Wiedziałem, że w celu przekazania informacji o rysunku do robota, będę musiał wybrać format, który nie zajmuje zbyt wiele pamięci. Zdecydowałem się na lekko zmodyfikowany podzbiór komend z szeroko wykorzystywanego języka PostScript. Udostępniłem użytkownikowi 3 komendy:

1. `x y lineto` - rysuje linię z obecnego miejsca do miejsca o współrzędnych (x, y).
2. `x y rlineto` - rysuje linię do miejsca oddalonego o wektor (x, y).
3. `phi r rlinerot` - obraca robota przeciwnie do ruchu wskazówek zegara o kąt phi i rysuje prostą linię o długości r.

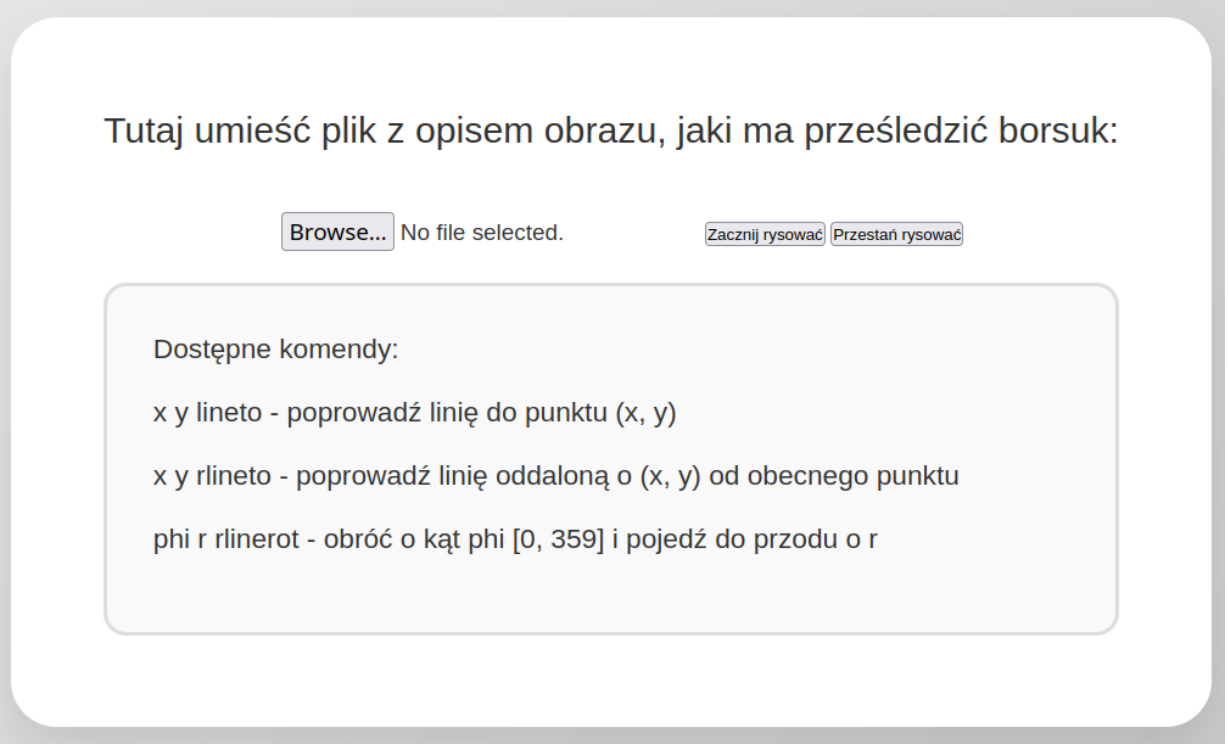
Z takich komend użytkownik tworzy plik, oddzielając je znakami nowej linii.

Pierwszym ciekawym zagadnieniem było to, jak przekształcić tak zapisane informacje na takie, które będą użyteczne w sterowaniu robotem. Stwierdziłem, że cały rysunek można podzielić na odcinki, a narysowanie odcinka składa się z dwóch faz: obrócenie robota w odpowiednim kierunku i narysowanie prostej linii.

Aby uzyskać takie informacje, stworzyłem klasę trzymającą dane o obecnym położeniu i kącie, o jaki obrócony jest robot. Przekształcając wektor, o jaki ma się przesunąć robot, z układu kartezjańskiego na biegunowy, łatwo można obliczyć, o jaki kąt trzeba obrócić robota i jak długą linię powinien narysować.

3 Interfejs użytkownika

Aby umożliwić użytkownikowi komfortową komunikację z robotem, rozszerzyłem dostępny kod odpowiedzialny za sterowanie borsukiem o kolejną stronę. Jest ona dostępna pod hi-

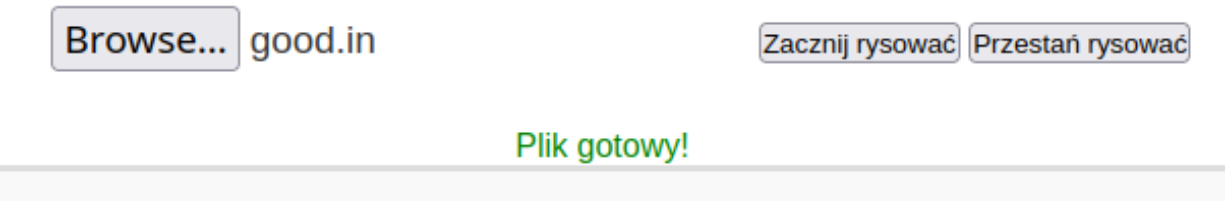


Rysunek 1: Interfejs użytkownika

perłączem opisanym jako '4'. Zaprojektowany przeze mnie interfejs użytkownika wygląda w następujący sposób.

Po naciśnięciu klawisza `Browse...` użytkownik wybiera plik zawierający opis rysunku. Jeśli plik jest prawidłowy, na ekranie ukazuje się odpowiednia informacja. Zawartość pliku 'good.in' to:

```
0 10 lineto
10 10 lineto
10 0 lineto
0 0 lineto
```



Rysunek 2: Informacja o poprawnym pliku

Po wykonaniu tych czynności użytkownik może nacisnąć przycisk `Zacznij rysować` i borsuk zacznie wykonywać rysunek.

4 Wykrywanie i obsługa błędów

Aplikacja weryfikuje wczytywany plik i wyświetla informacje o błędzie. Tylko wczytanie poprawnego pliku spowoduje, że naciśnięcie przycisku `Zacznij rysować` będzie miało efekt. W pliku wejściowym wykrywane są następujące błędy:

- Niepoprawna ilość argumentów
- Nierozpoznane polecenie
- Niepoprawne wartości liczbowe

Przykładowy niepoprawny plik 'bad1.in':

```
90 3 rlineto
a3 4 lineto
```

Rysunek 3: Komunikat o niepoprawnym pliku

W przypadku, gdy stanie się coś, co będzie wymagało przerwania rysunku, przycisk `Przestań rysować` zatrzymuje wykonywanie rysunku na obecnej komendzie. Skutkuje to utratą obecnego postępu - aby rozpocząć rysowanie od nowa, trzeba ponownie wczytać poprawny plik.

5 To co się udało i napotkane problemy

Wczytywanie pliku oraz interpretowanie poszczególnych instrukcji zadziałało bardzo dobrze. Podczas testów nie udało mi się wykryć błędów i wydaje mi się, że wszystkie instrukcje podawane robotowi są poprawne.

Jednak dużym problemem okazało się przełożenie tych instrukcji na ruch robota. Najwygodniejszym miejscem do montażu pisaka, tak aby leżał on na osi symetrii robota, był przód pojazdu. Niestety z tego powodu pisak był oddalony o duży dystans od osi obrotu

pojazdu. To oznaczało, że obrót robota wokół własnej osi nie będzie dobrym przybliżeniem obrotu robota wokół osi pisaka. Ze względu na to, że nie byłem pewien, jak dobrze będzie działać mój projekt, zdecydowałem się tylko na tymczasowe przymocowanie pisaka do pojazdu. Postanowiłem skupić się na obsługiwaniu obrotów, będących wielokrotnością $\pi/2$.

W tym celu chciałem skalibrować taką sekwencję, gdzie robot będzie skręcał w jedną stronę, jadąc do przodu, i w przeciwną, jadąc do tyłu. Miałem nadzieję, że złożenie kilku takich ruchów spowoduje, że robot obróci się wokół osi pisaka. Niestety, ze względu na wysoki próg mocy, przy którym koło zaczyna się obracać, bardzo ciężkim zadaniem była kalibracja odpowiedniej mocy, tak aby koło, które ma poruszać się wolniej, poruszało się odpowiednio szybko. Gdy wartość wypełnienia była zbyt mała, robot po prostu obracał się wokół osi nieruchomego koła. Natomiast gdy wypełnienie było wystarczająco duże, robot poruszał się z dość dużą prędkością i ciężko było skalibrować go tak, aby był w stanie wykonać precyzyjny rysunek.

Kolejnym problemem był kontakt pisaka z kartką. Pomimo faktu, iż starałem się, aby pisak miał minimalny kontakt z kartką, wciąż wpływał on na ruch robota. Był to kolejny czynnik sprawiający, że ruch robota był ciężki do przewidzenia. Ponadto był też składową problemu, gdzie poszczególne koła traciły przyczepność, zmieniając trajektorię jazdy.

Powyższe czynniki sprawiają, że bardzo ciężko jest uzyskać precyzyjny rysunek - każda niedokładność w wykonywanym manewrze nakłada się na kolejne manewry.

6 Możliwe usprawnienia

Pierwszym usprawnieniem, jakie przychodzi do głowy, jest stworzenie dobrego elementu trzymającego pisak oraz mechanizmu podnoszenia i opuszczania pisaka. Niestety mam wrażenie, że takie usprawnienia z pewnością poprawiłyby dokładność wykonywanych rysunków, lecz nie byłyby w stanie naprawić głównego problemu - zbyt dużej prędkości minimalnej silników.

Z tego powodu używanie tego robota jako rysownika jest dość nieporęczne. W celu wykonania nawet bardzo prostego rysunku potrzeba bardzo dużej powierzchni roboczej. Duża prędkość obrotowa kół sprawia również, że ruchy robota są mniej dokładne, a koła często tracą przyczepność (nie pomaga fakt, że poruszają się na papierze), co znacząco pogarsza jakość rysunku.

Najlepszym usprawnieniem dla tego projektu byłoby zbudowanie robota, który wykonuje ruchy powoli, lecz z większą dokładnością, i wyposażenie go w przyczepne koła.

Na szczęście napisany kod działa głównie po stronie interfejsu użytkownika i wysyła do robota jedynie wartości, z jakimi mają się obracać silniki. Jest on więc całkiem przenośny.

7 Film z działania

Nagranie znajduje się pod tym linkiem. [1]

Robot miał wykonać rysunek kwadratu. Mam nadzieję, że w miarę widać, że faktycznie wykonuje on sekwencję:

- krótka prosta linia
- obrót w prawo o $\pi/2$
- krótka prosta linia
- obrót w prawo o $\pi/2$
- krótka prosta linia
- obrót w prawo o $\pi/2$
- krótka prosta linia

Niestety, ze względu na opisane przeze mnie wcześniej problemy, mimo faktu, że rysowane linie są całkiem proste (mam na myśli tylko te linie, które nie były rysowane podczas zwrotów), obroty robota były obarczone bardzo dużym błędem. Oczywiście, gdyby linie proste były znacząco dłuższe niż te, które robot nakreśla podczas obrotów, rysunek mógłby wyglądać lepiej. Niestety zorganizowanie tak dużej przestrzeni roboczej jest bardzo nieporęczne.

8 Podsumowanie

Chociaż nie do końca udało się spełnić to, co miałem w głowie na początku projektu, jestem z niego zadowolony. Udało mi się po drodze nauczyć kilku ciekawych rzeczy. Co prawda stworzony projekt nie posłuży raczej do narysowania precyzyjnego rysunku, ale dzięki niemu można stworzyć schemat poruszania robotem, który będzie całkiem powtarzalny, lub stworzyć rysunek abstrakcyjny.

[1] Link do nagrania: `https://drive.google.com/file/d/174hAblXnFykchKyTH_9HKrq8cTLBl-A4/view?usp=sharing`