

SVEUČILIŠTE U SPLITU  
FAKULTET ELEKTROTEHNIKE STROJARSTVA I BRODOGRADNJE

PROGRAMIRANJE MOBILNIH ROBOTA I LETJELICA

**POLOLU**

MATEO RADIĆ

SIJEČANJ, 2022.

# Sadržaj

1	Uvod.....	1
2	Postavljanje okruženja .....	2
2.1	Instalacija potrebnih paketa.....	2
2.2	Rad s bibliotekama .....	5
3	Rekonstruiranje rute uz pomoć enkodera .....	7
3.1	Zapis podatak enkodera.....	7
3.2	Kretnja robota i rekonstrukcija putanje.....	8
4	Praćenje krivulje uz pomoć enkodera .....	10
4.1	Prikupljanje osnovnih podataka robota .....	10
4.2	Programska inicijalizacija .....	11
4.3	Pronalazak prve točke .....	12
4.3.1	Problem s dolaskom u ciljnu točku.....	15
4.4	Rješenje pravocrtnog gibanja .....	18
4.5	Crtanje kvadra s robotom .....	21
5	Zaključak.....	22

# 1 Uvod

Cilj seminara je upoznati se s Pololu robotom i njegovim funkcionalnostima. Kroz iteraciju s enkoderima bit će rekonstruirane rute koju je robot prošao samo iščitavajući vrijednosti koje je robot zapisao prilikom inicijalnog prolaska.

Također, u seminaru je objašnjen način određivanja koordinata robota uz pomoć enkodera, te praćenje unaprijed zadanih točaka krivulje.

Cijeli seminar je popraćen isječcima kodova kako bi olakšalo shvaćanje implementacije u realnim primjerima.

## 2 Postavljanje okruženja

U ovoj cjelini detaljno je objašnjeno postavljanje okruženja za početak rada s robotom.

### 2.1 Instalacija potrebnih paketa

Kako bi započeli programiranje na pololu 3pi+ robotu potrebno je ispratiti nekoliko jednostavnih koraka:

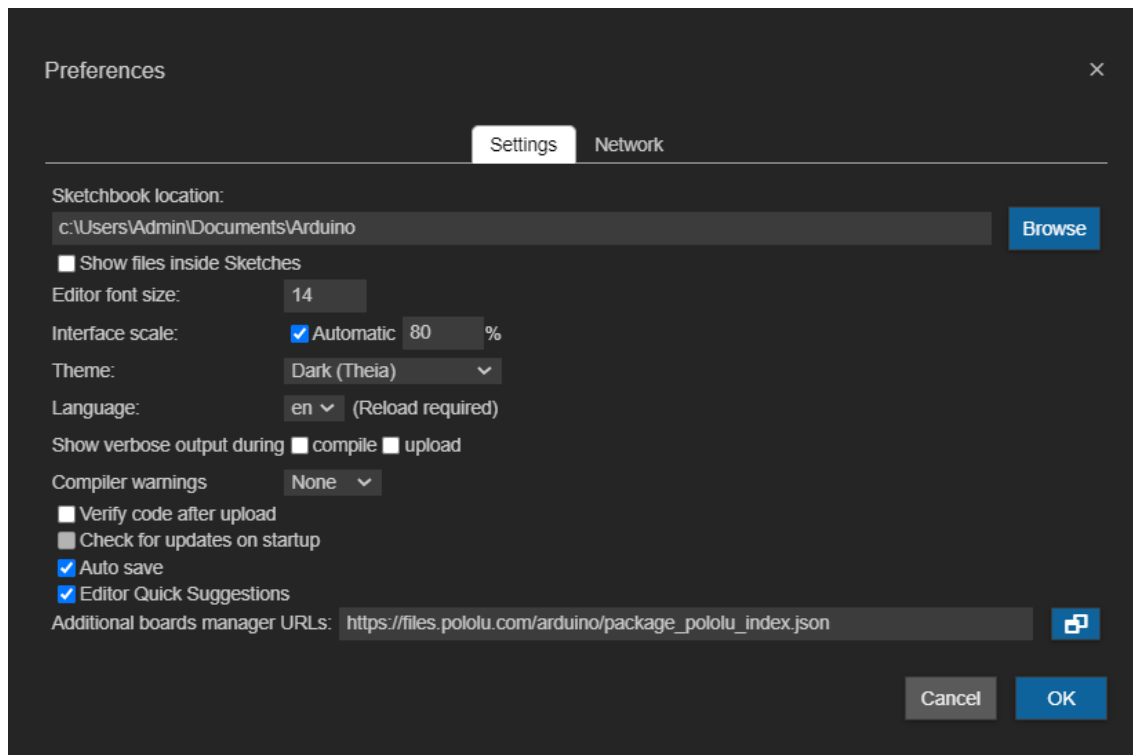
1. Preuzimanje i instaliranje Arduino IDE,
2. U Arduino IDE potrebno je otvoriti

**File -> Preferences**

3. U otvorenom prozoru potrebno je dodati link

[https://files.pololu.com/arduino/package\\_pololu\\_index.json](https://files.pololu.com/arduino/package_pololu_index.json)

u „Additional Boards Manager URLs“

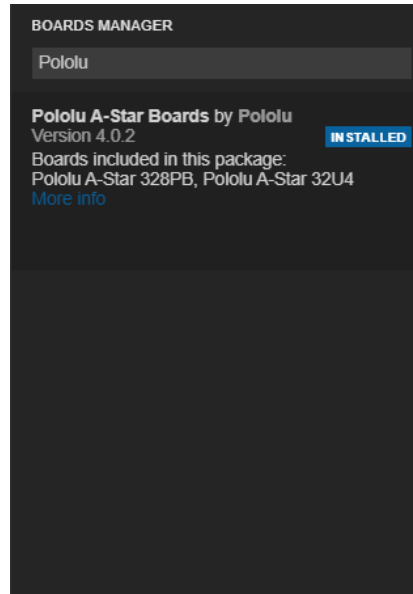


te potvrditi s OK

4. Potrebno je odabrati 32U4 ploču koja pokreće robota, stoga je potrebno instalirati. U izborniku odabiremo

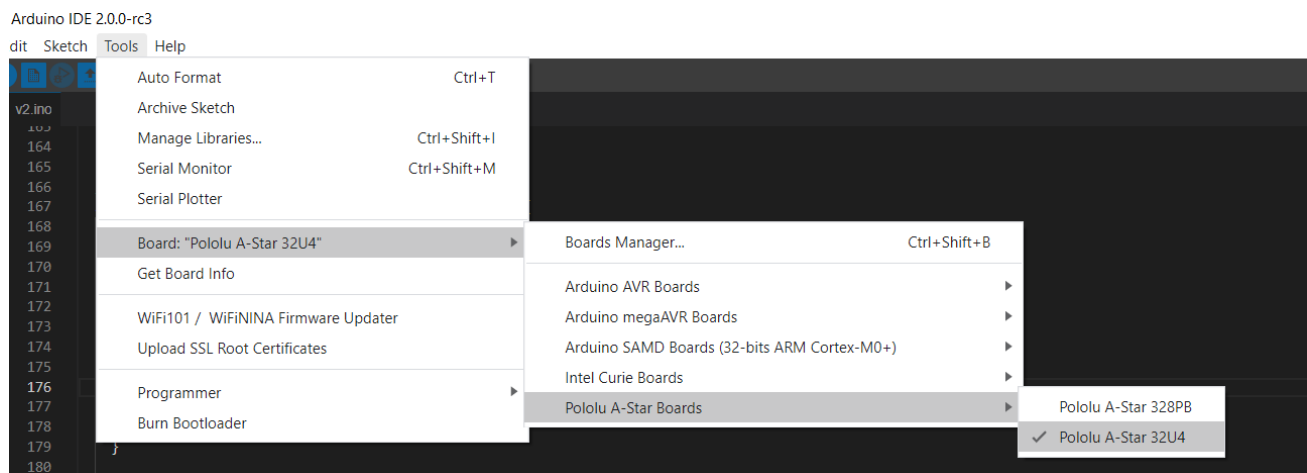
### Tools -> Board Manager

te pretražujemo Pololu i instaliramo pakete



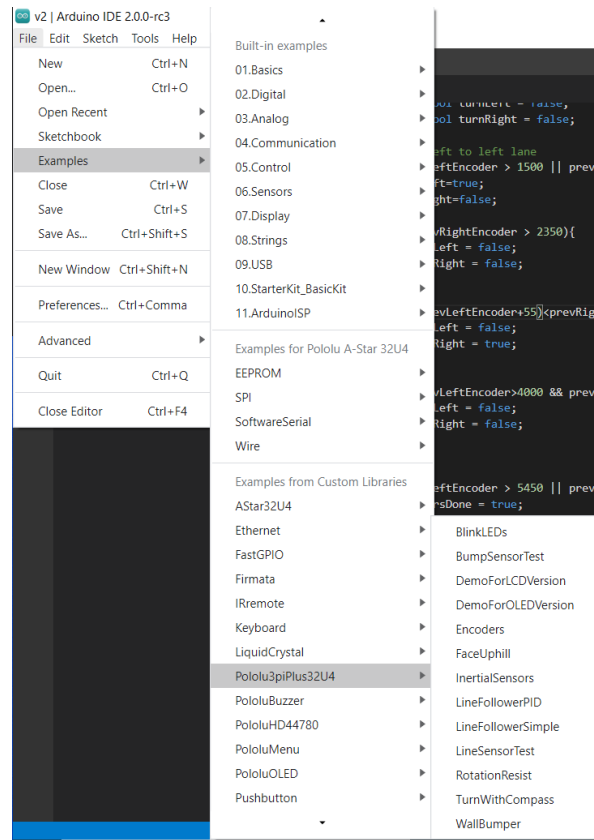
5. U prethodnom koraku smo instalirali pakete za našeg robota „32U4“, sada je trebamo odabrati. Idemo na

### Tools -> Board -> Pololu A-Star Boards -> Pololu A-Star 32U4

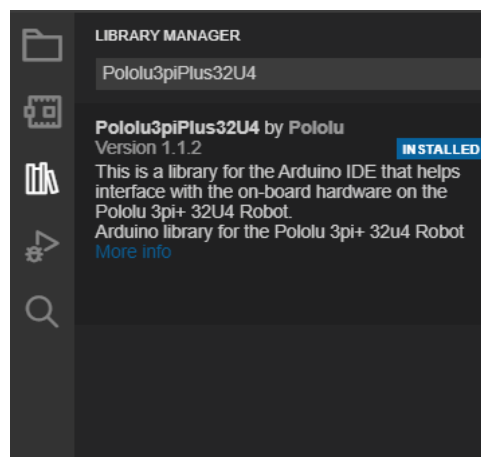


Sada je okruženje postavljeno i dostupni su nam razni primjeri specifični za našeg robota. Pronalazimo ih u izborniku

### Examples -> Pololu3piPlus32U4



6. Kako bi pojednostavljeno mogli koristiti sve funkcionalnosti robota s predefiniranim funkcijama potrebno je instalirati robotovu biblioteku Pololu3piPlus32U4.



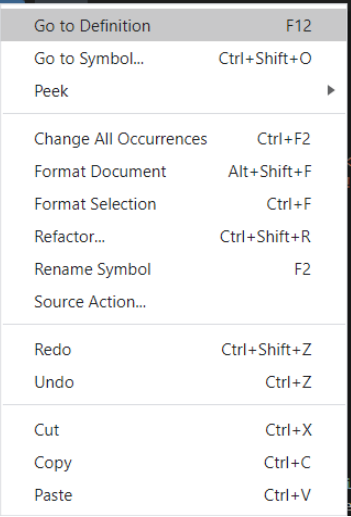
## 2.2 Rad s bibliotekama

Nakon instalacije paketa dostupne su nam predefinirane funkcije robota samo ih je potrebno uključiti u naš program.

```
v2.ino
1  #include <Pololu3piPlus32U4.h>
2
3  using namespace Pololu3piPlus32U4;
4
5  Encoders encoders;
6  Buzzer buzzer;
7  Motors motors;
8  ButtonA buttonA;
9  ButtonC buttonC;
10 ButtonB buttonB;
11 LCD display;
12
13
```

Pregled dostupnih funkcija možemo pogledati odlaskom na definiciju same klase

```
1  #include <Pololu3piPlus32U4.h>
2
3  using namespace Pololu3piPlus32U4;
4
5  Encoders encoders;
6  Buzzer buzzer;
7  Motors motors;
8  ButtonA buttonA;
9  ButtonC buttonC;
10 ButtonB buttonB;
11 LCD display;
12
13
14 const int ...
15 const int ...
16
17 int ...
18 int ...
19 unsigned ...
20
21 int ...
22
23 char ...
24
25 int ...
26 int ...
27
28 void ...
29
30
```



Dostupne su nam funkcije koje se nalaze u pristupnom modifikatoru „public“ unutar klase.

```
16 class Motors
17 {
18     public:
19
20         /// \brief Flips the direction of the left motor.
21         ///
22         /// You can call this function with an argument of \c true if the left motor
23         /// of your 3pi+ was not wired in the standard way and you want a
24         /// positive speed argument to correspond to forward movement.
25         ///
26         /// \param flip If true, then positive motor speeds will correspond to the
27         /// direction pin being high. If false, then positive motor speeds will
28         /// correspond to the direction pin being low.
29         ///
30         static void flipLeftMotor(bool flip);
31
32         /// \brief Flips the direction of the right motor.
33         ///
34         /// You can call this function with an argument of \c true if the right
35         /// motor of your 3pi+ was not wired in the standard way and you want a
36         /// positive speed argument to correspond to forward movement.
37         ///
38         /// \param flip If true, then positive motor speeds will correspond to the
39         /// direction pin being high. If false, then positive motor speeds will
40         /// correspond to the direction pin being low.
41         static void flipRightMotor(bool flip);
42
43         /// \brief Sets the speed for the left motor.
44         ///
45         /// \param speed A number from -400 to 400 representing the speed and
46         /// direction of the left motor. Values of -400 or less result in full
47         /// speed reverse, and values of 400 or more result in full speed forward.
48         static void setLeftSpeed(int16_t speed);
```



### 3 Rekonstruiranje rute uz pomoć enkodera

U ovom poglavlju robot će ponoviti radnju koristeći samo zapis iz enkodera. Radnja koju će robot obavljati je prestrojavanje u lijevu traku.



#### 3.1 Zapis podatak enkodera

Kako bi dobili trenutno očitavanje enkodera dostupne su nam funkcije:

- `encoders.getCountsLeft();`
- `encoders.getCountsRight();`

koristeći predefinirane funkcije očitavamo trenutnu vrijednost i uspoređujemo s prošlim očitanjem enkodera svakih 100 ms. Ukoliko je vrijednost različita zapisujemo novu vrijednost enkodera u polje zajedno s vremenom kad je nastupilo očitavanje nove vrijednosti enkodera.

```
193 //Check is there any change happening
194 if(countsLeft != prevLeftEncoder || countsRight != prevRightEncoder){
195     prevLeftEncoder = countsLeft;
196     prevRightEncoder = countsRight;
197     updateEncoders(millis());
198 }
```

Stoga će polje zapisa lijevog i desnog enkodera izgledati

`left = [{value: xx, timestamp: xxxxxx}, {value: xx, timestamp: xxxxxx}, ...]`

`right = [{value: xx, timestamp: xxxxx}, {value: xx, timestamp: xxxxx}, ...]`

Kad želimo završiti kretanju potrebno je pritisnuti B botun

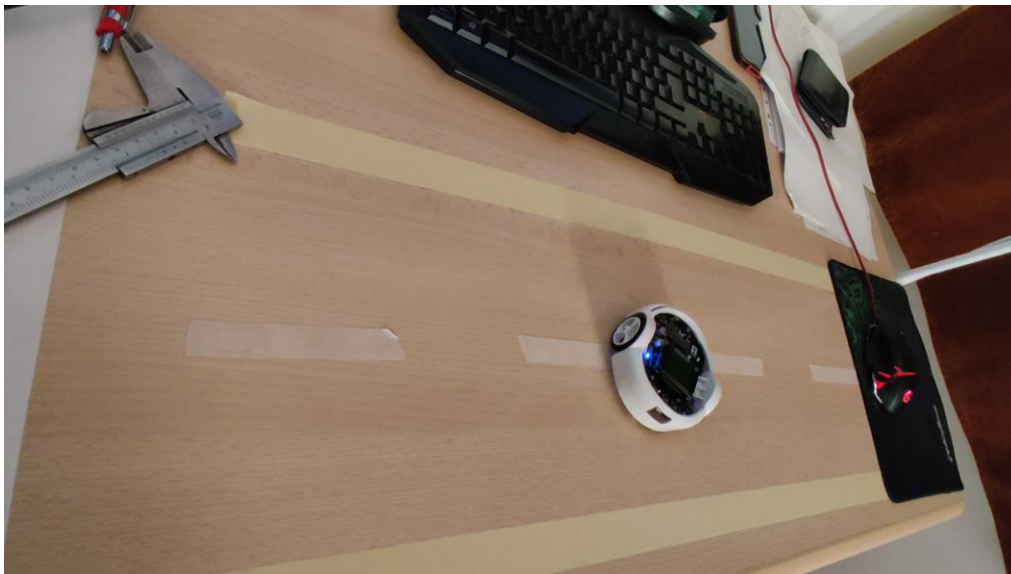
```
245     if(buttonB.isPressed()){
246         //showEncodersReport();
247         displayEncoderCounts(0, 0);
248         delay(1000);
249         encoders.getCountsAndResetLeft();
250         encoders.getCountsAndResetRight();
251         followEncoders();
252         encodersDone=true;
253     }
```

Poništi se očitavanje enkodera te pozove funkcija „followEncoders()“ za ponavljanje radnje koje su zapisali enkodери.

### 3.2 Kretanja robota i rekonstrukcija putanje

Robot započinje kretanju:

- unaprijed,
- skrene lijevo i nastavi se kretati unaprijed dok stigne do sredine lijeve trake
- zatim skrene desno za istu vrijednost koliko je skrenuo ulijevo (kako bi se nastavio gibati paralelno s smjerom gibanja u desnoj traci)
- nastavi se gibati ravno do kraja trake



Kad robot stigne do kraja putanje koju želimo rekonstruirati potrebno je pritisnuti botun B koji pokreće iščitavanje zapisa enkodera.

```
52 void followEncoders(){
53     int16_t countsLeft = encoders.getCountsLeft();
54     int16_t countsRight = encoders.getCountsRight();
55
56     for(int i=0; i<filledIndex; i++){
57         int16_t countsLeft = encoders.getCountsLeft();
58         int16_t countsRight = encoders.getCountsRight();
59         displayEncoderCounts(countsLeft, countsRight);
60
61         if(countsLeft<leftEncoder[i] ){
62             motors.setLeftSpeed(175);
63         }else{
64             motors.setLeftSpeed(0);
65         }
66
67         if(countsRight<rightEncoder[i] ){
68             motors.setRightSpeed(175);
69         }else{
70             motors.setRightSpeed(0);
71         }
72
73         delay(120);
74     }
75 }
```

Ukoliko je trenutna vrijednost enkodera manja od vrijednosti zapisane u polju iz zadane kretnje postavi motor da se kreće unaprijed te prati vrijednost enkodera.

Kad se dostigne vrijednost zapisa enkodera, čekaj iduću vrijednost kako bi je mogao „sustići“ i na taj način rekonstruirati kretnju zapisanu enkoderima.

NAPOMENA: Konstanta je brzina prilikom gibanja robota.

## 4 Praćenje krivulje uz pomoć enkodera

U ovom poglavlju uz pomoć zapisa enkodera odrediti ćemo lokaciju robota u prostoru i na taj način usmjeriti robota da giba po točno definiranim točkama.

### 4.1 Prikupljanje osnovnih podataka robota

Kako bi mogli odrediti položaj robota u prostoru

$$\begin{aligned}x' &= x + D_c \cos(\phi) \\y' &= y + D_c \sin(\phi) \\ \phi' &= \phi + \frac{D_r - D_\ell}{L}\end{aligned}$$

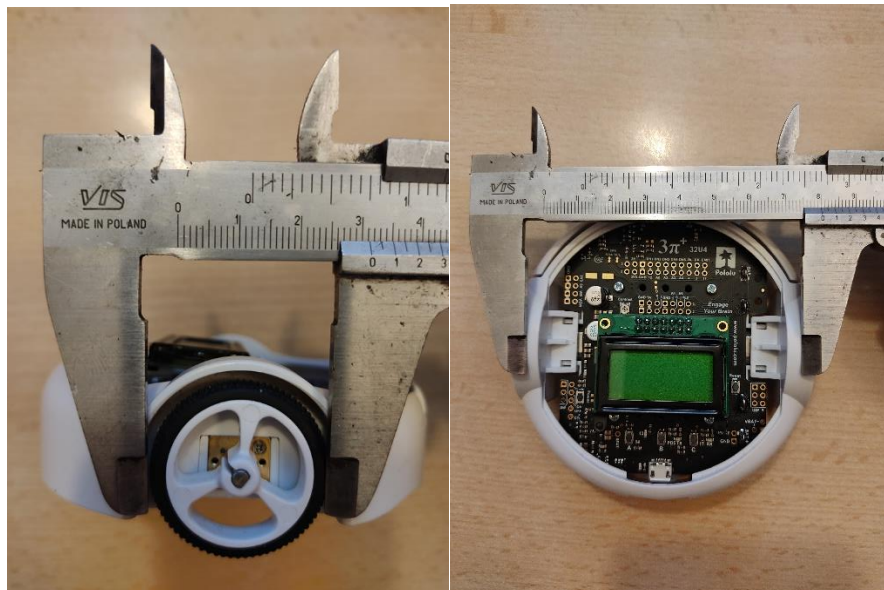
Potrebne su nam opće informacije o robotu poput radijusa kotača - R, te udaljenosti između kotača – L. To nam je potrebno kako bi mogli odrediti udaljenost koju je prevalio svaki kotač.

$$\begin{aligned}D_\ell &= 2\pi R \frac{\Delta tick_\ell}{N} \\D_r &= 2\pi R \frac{\Delta tick_r}{N} \\D_c &= \frac{D_r + D_\ell}{2}\end{aligned} \quad \left. \vphantom{\begin{aligned}D_\ell \\D_r \\D_c\end{aligned}} \right\} dt$$

Empirijski je određeno da enkoder izbroji:

1. 908 tickova za puni okret lijevog kotača,
2. 906 tickova za puni okret desnog kotača,

Te radijus kotača sadrži 16mm, razmak između kotača je 80mm.



## 4.2 Programska inicijalizacija

Najprije je potrebno definirati funkcije koje će nam zasigurno trebati:

1. Izračun prijedene udaljenosti kotača

```

43 float DL_handler(){
44     int16_t countsLeft = encoders.getCountsLeft();
45     float dl = 2*3.14159*R*(countsLeft - prevLeftTick)/900;
46     prevLeftTick = countsLeft;
47     return dl;
48 }
49
50 float DR_handler(){
51     int16_t countsRight = encoders.getCountsRight();
52     float dr = 2*3.14159*R*(countsRight - prevRightTick)/906;
53     prevRightTick = countsRight;
54     return dr;
55 }
56
57 float DC_handler(){
58     return (DR_handler() + DL_handler())/2;
59 }

```

2. Izračun trenutnih koordinata i kuta robota

```

86 void calculateNewRobotCordinate(){
87     currentY = currentY + DC_handler()*sin(currentAngle);
88     currentX = currentX + DC_handler()*cos(currentAngle);
89 }
90
91 float calculateCurrentAngle(){
92     currentAngle = currentAngle + ((DR_handler() - DL_handler())/L);
93     return currentAngle;
94 }

```

### 3. Izračun udaljenosti robota do željene točke, te kut do željene točke

```

96 void calculateDistanceToGoal(){
97     float y = pow((currentY - desiredPointY),2);
98     float x = pow((currentX - desiredPointX),2);
99     distance = sqrt(x+y);
100 }
101
102 void calculateGoalAngle (int newY, int newX ){
103     desiredAngle = atan2(newY - currentY, newX - currentX);
104     desiredPointY = newY;
105     desiredPointX = newX;
106     rightDirection=false;
107 }

```

atan2 funkcija osigurava da vraćena vrijednost bude između pi i -pi.

### 4.3 Pronalazak prve točke

Robota postavljamo u središte koordinatnog sustava (currentX, currentY) s kutom 0 (currentAngle).

```

32
33 int prevLeftTick = 0;
34 int prevRightTick = 0;
35
36 float currentX = 0;
37 float currentY = 0;
38 float currentAngle = 0;
39
40 const int R = 16;
41 const int L = 80;

```

te definiramo točke koje robot treba „pronaći“.

```
21  int pathYCoordinates [GOAL_DOTS_NUMBER] = {50};  
22  int pathXCoordinates [GOAL_DOTS_NUMBER] = {-50};
```

u našem slučaju to je samo jedna točka (-50, 50).

Prilikom pokretanja robota u setup() funkciji

```
113  void setup()  
114  {  
115      calculateGoalAngle(pathYCoordinates[0], pathXCoordinates[0]);  
116      currentGoalDot++;  
117      displayEncoderCounts(0,0);  
118      delay(2000);  
119  }  
120
```

uzimamo koordinate prve točke te računamo kut od trenutne pozicije robota do željene točke (u našem slučaju (-50,50)).

Sad kad je postavljen željeni kut kojim se robot treba kretati kako bi došao do zadane točke potrebno je rotirati robota dok se trenutni kut robota približno ne podudara s željenim kutem točke u kojem smjeru robot treba ići.

```

121 void loop()
122 {
123     static uint8_t lastDisplayTime;
124     static uint8_t displayErrorLeftCountdown = 0;
125     static uint8_t displayErrorRightCountdown = 0;
126
127     calculateCurrentAngle();
128
129
130     if((currentAngle-0.01 < desiredAngle && desiredAngle < currentAngle+0.01) || rightDirection){
131         if(!rightDirection){
132             prevLeftTick = 0;
133             encoders.getCountsAndResetLeft();
134             prevRightTick = 0;
135             encoders.getCountsAndResetRight();
136         }
137         rightDirection = true;
138         calculateNewRobotCordinate();
139         distanceCheckHandler();
140
141         if(goalReached){
142             motors.setSpeeds(0, 0);
143             delay(10000);
144         }else{
145             motors.setSpeeds(120, 120);
146         }
147     }else if (currentAngle < desiredAngle){
148         motors.setSpeeds(-100, 100);
149     }else if (currentAngle > desiredAngle){
150         motors.setSpeeds(100, -100);
151     }
152 }

```

Na liniji 147 i 149 radimo provjeru je li željeni kut manji ili veći od trenutnog kako bi pravilno rotirali robota prema željenom kutu.

Ako robot se robot okrenuo prema ciljnom položaju (-50, 50). Spreman je na početak gibanja unaprijed.

Kad se robot počeo kretati došlo je do promjene zapisa enkodera i potrebno je izračunati nove koordinate robota, te također i novu udaljenost robota od trenutne pozicije do zadane ciljne točke.

```

61 void distanceCheckHandler(){
62     calculateDistanceToGoal();
63
64     // Serial.println(distance);
65     if(distance < 10){
66         if(currentGoalDot < GOAL_DOTS_NUMBER){
67             calculateGoalAngle(pathYCoordinates[currentGoalDot], pathXCoordinates[currentGoalDot]);
68             currentGoalDot++;
69             rightDirection=false;
70
71         }else{
72             // Serial.println(currentX);
73             // Serial.println(currentY);
74             // Serial.println();
75             goalReached = true;
76         }
77     }
78 }

```

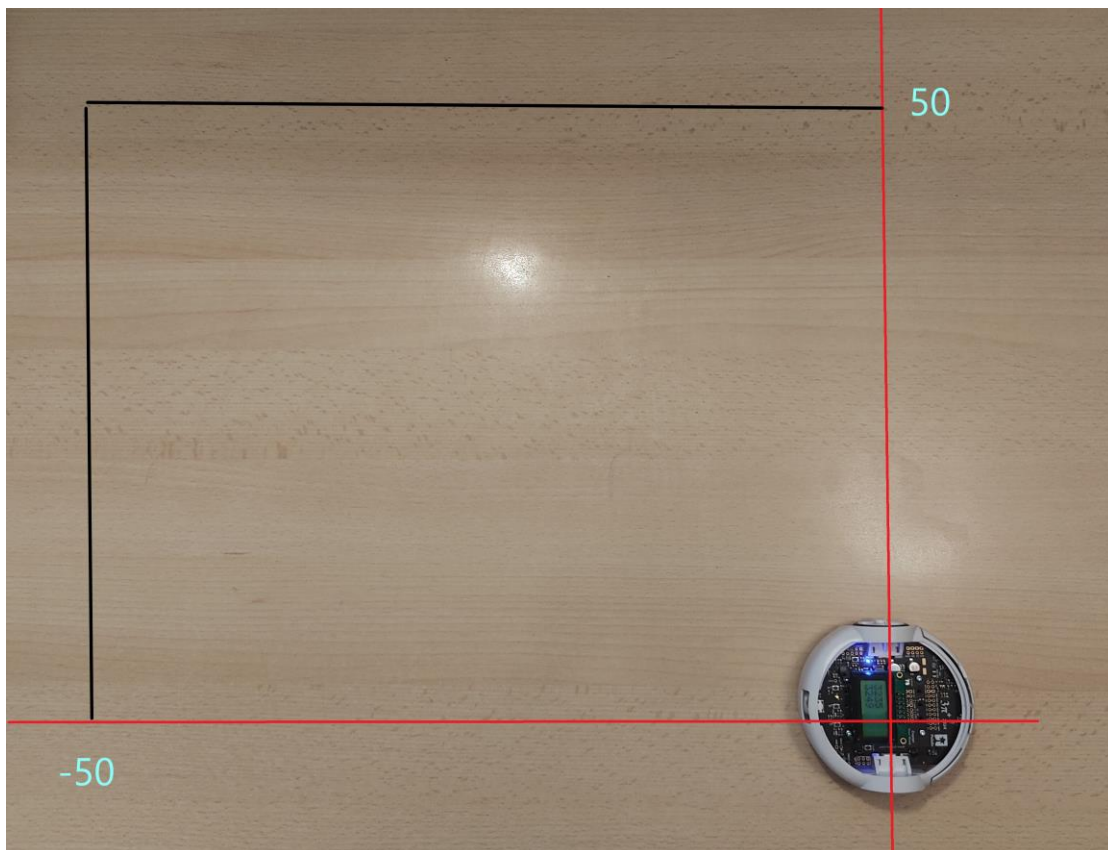


Ukoliko je udaljenost robota manja od 10 smatra se da je robot stigao u ciljnu točku te je spreman za postavljanje nove ciljne točke ili u našem slučaju više nemamo točki te postavljamo flag da smo stigli do cilja (goalReached – globalna varijabla ukoliko je true postavlja robota u mirovanje).

#### 4.3.1 Problem s dolaskom u ciljnu točku

Prethodno opisani način dolaska do ciljne točke uvelike ovisi napunjenosti baterije, te točnosti i preciznosti istog očitavanja s enkodera uslijed više pokušaja.

Postavimo situaciju da se robot nalazi u točki (0,0) s trenutnim kutem 0 u odnosu na x os.



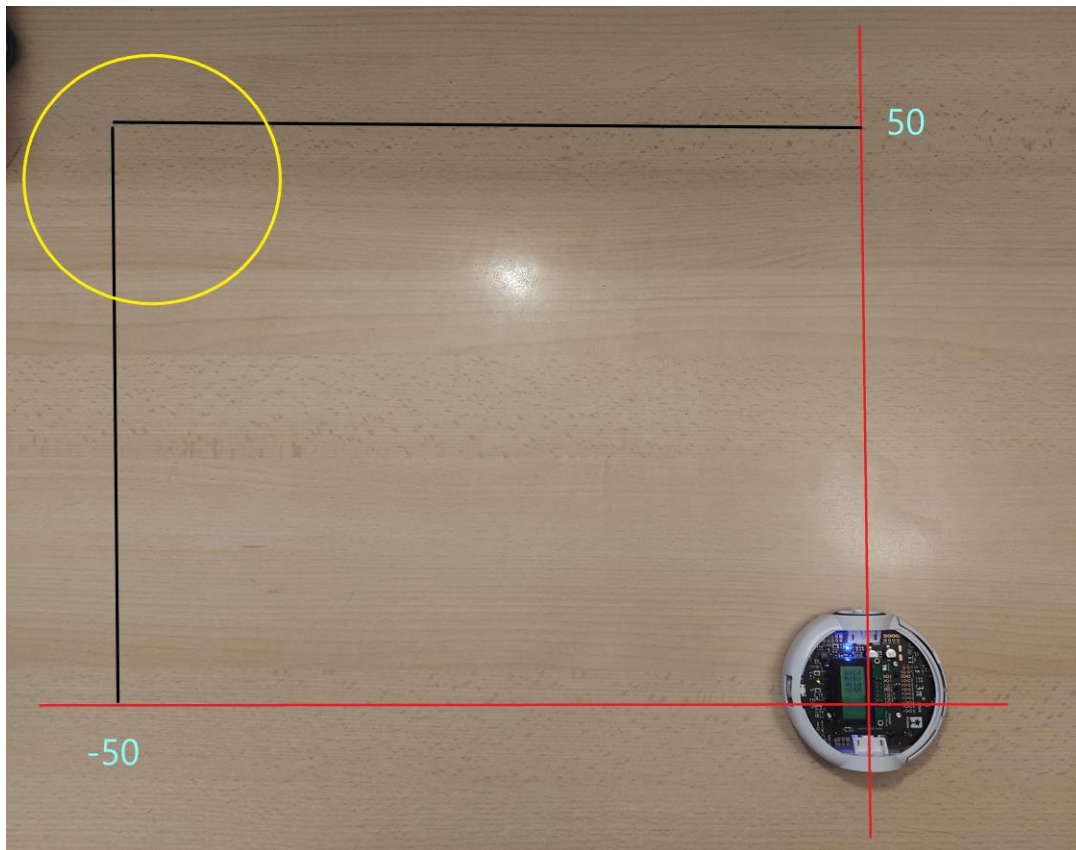
Dakle, naša željena točka je (-50,50). Robot se uz pomoć očitavanja enkodera okrene prema točki te krene prema njoj. Kako bi mogao doseći cilj i zadovoljiti uvjet

```

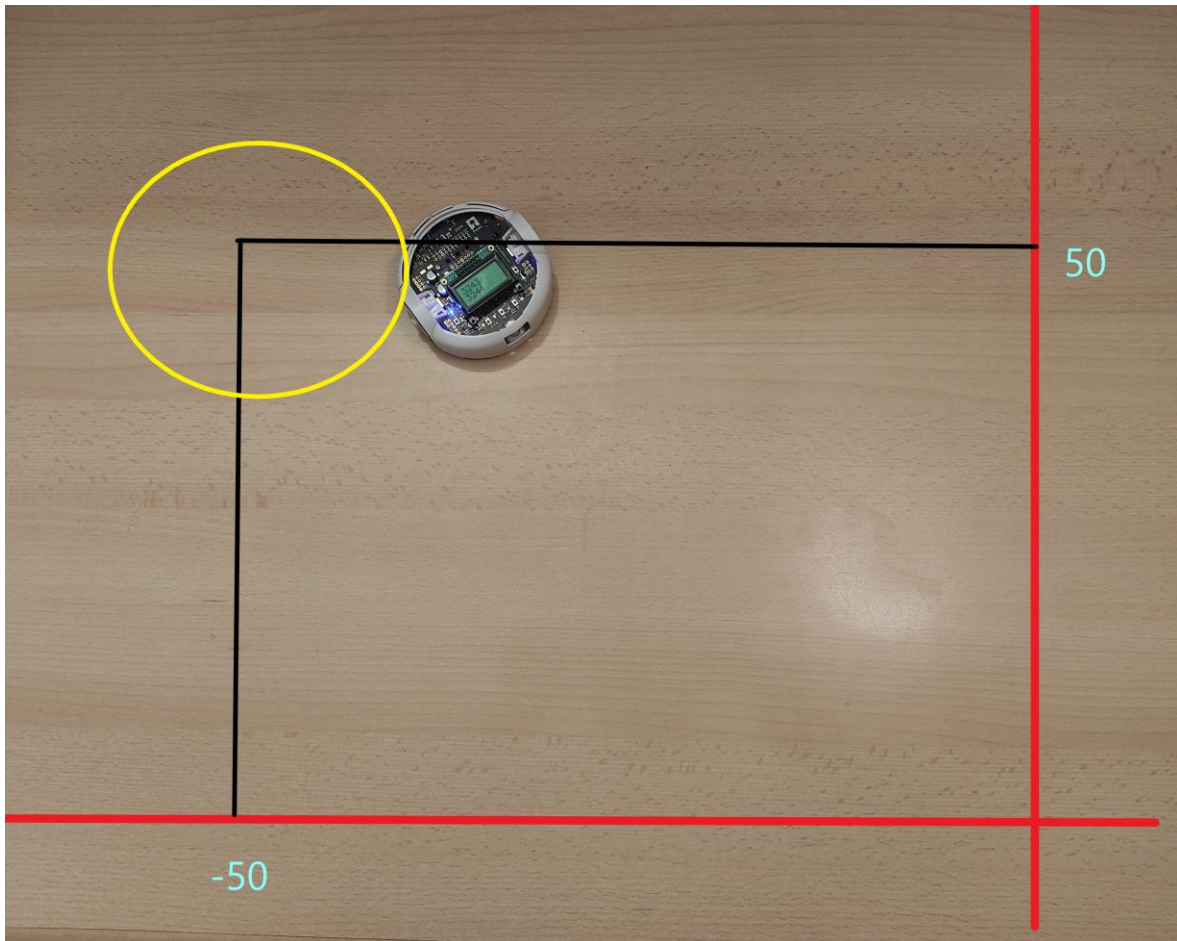
61 void distanceCheckHandler(){
62     calculateDistanceToGoal();
63
64     // Serial.println(distance);
65     if(distance < 10){
66         if(currentGoalDot < GOAL_DOTS_NUMBER){
67             calculateGoalAngle(pathYCoordinates[currentGoalDot], pathXCoordinates[currentGoalDot]);
68             currentGoalDot++;
69             rightDirection=false;
70         }else{
71             // Serial.println(currentX);
72             // Serial.println(currentY);
73             // Serial.println();
74             goalReached = true;
75         }
76     }
77 }
78 }

```

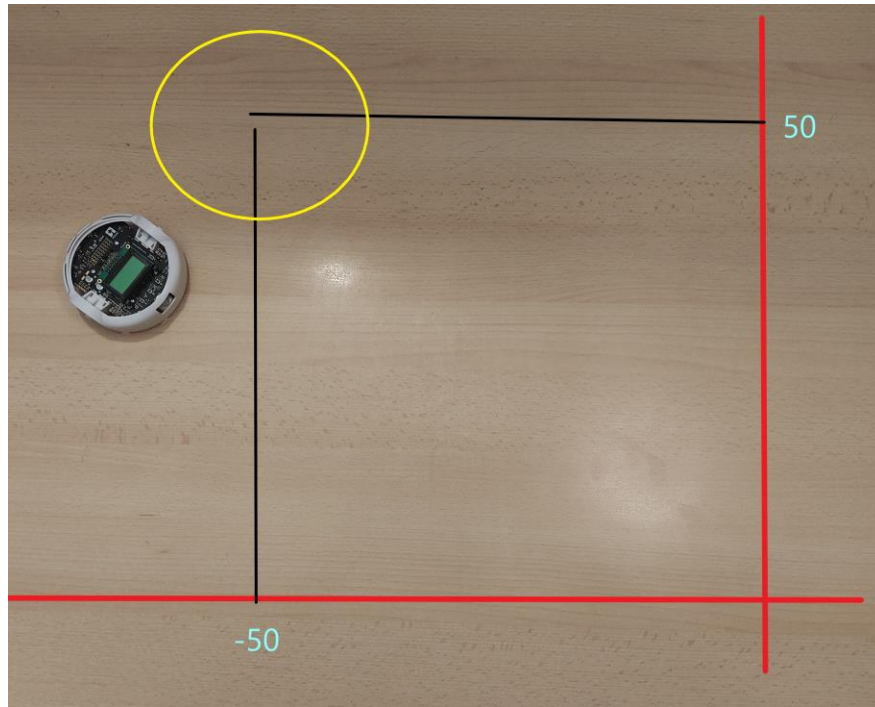
Dodali smo proširenje cilja, tj. Čim se robot nalazi na udaljenosti manjoj od 10 do ciljne točke uvjet je zadovoljen i smatra se da je robot dosegao svoju ciljnu točku.



Problem ovog pristupa je taj što robot nikad neće dostići ciljnu točku, već približne točke koje ovise o paddingu kojeg smo dodali oko ciljne točke



I s obzirom da uvelike ovisimo o napunjenosti baterija, startnom odazivu elektromotora u velikom broju pokušaja se dogodi da robot u potpunosti promaši ciljnu točku. Tako da se dobivanje preciznosti dolaska do ciljne točke smanjivanjem radijusa oko ciljne točke odbacuje.



Robot kad prođe pored točke nema u sebi ugrađenu regulaciju povratne veze za dohvaćanje ciljne točke, stoga samo nastavi daljnje gibanje po pravcu do beskonačnosti. Što je i razumljivo jer robot se okrenuo prema točki i sljedeći korak za dolazak to cilja je samo gibanje prema naprijed.

#### 4.4 Rješenje pravocrtnog gibanja

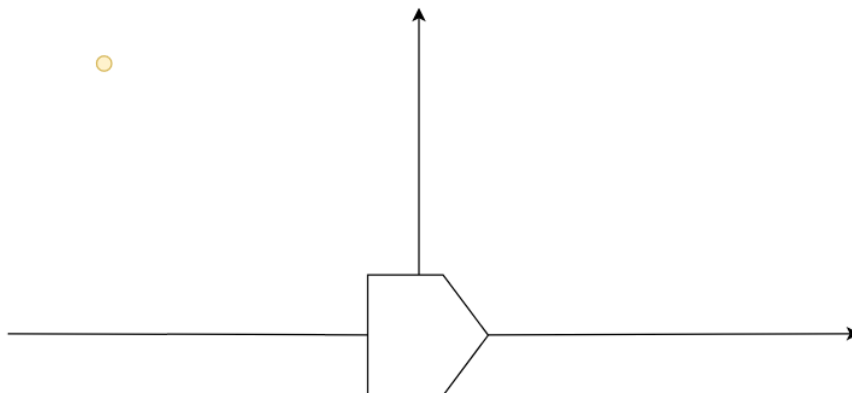
Kako znamo da je pomak kotača

$$\begin{aligned}
 D_l &= 2\pi R \frac{\Delta tick_l}{N} \\
 D_r &= 2\pi R \frac{\Delta tick_r}{N}
 \end{aligned}
 \left. \vphantom{\begin{aligned} D_l &= 2\pi R \frac{\Delta tick_l}{N} \\ D_r &= 2\pi R \frac{\Delta tick_r}{N} \end{aligned}} \right\} dt$$

$$D_c = \frac{D_r + D_l}{2}$$

a konstante su nam poznate možemo izračunati potreban broj tickova za prevladavanje potrebne distance.

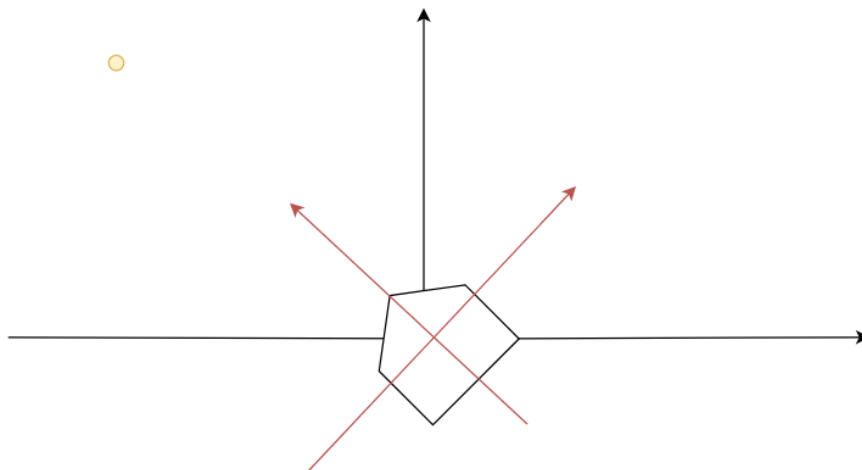
Pretpostavimo da se robot nalazi u točki  $(0, 0)$  s kutom  $0$  u odnosu na os  $x$ . te želi stići u ciljnu točku  $(-300, 300)$ .



Robot do ciljne točke dolazi u dva koraka:

1. Okreni se prema ciljnoj točki
2. Kreći se u naprijed dok ne prijeđeš udaljenost do točke

Dakle u prvom koraku izračunajmo kut do ciljne točke, te okrenimo robota prema ciljnoj točki. Sada kada je robot okrenut prema ciljnoj točki s njom zatvara kut od  $0$  stupnjeva, i giba se samo unaprijed kao bi došao do ciljne točke.



Pozitivna X-os je u smjeru kretanja robota, a kako je kut koji robot zatvara s ciljnom točkom 0 potrebno je izračunati koliki broj tickova treba izbrojati enkoder kako bi robot prevalio udaljenost do ciljne točke.

$$\Delta \text{ tick} = D_C * N / 2 * \pi * R$$

Gdje je  $D_C$  udaljenost koju kotači moraju preći do ciljne točke, određujemo jednostavnim pozivom funkcije.

```
104 void calculateDistanceToGoal(){
105     float y = pow((currentY - desiredPointY),2);
106     float x = pow((currentX - desiredPointX),2);
107     distance = sqrt(x+y);
108 }
```

Programski izvodimo pravocrtno gibanje do točke na sljedeći način.

```
64 void reachDistance(){
65     float tickGoal = (distance*907)/(2*3.14159*R);
66     int encoders_count = 0;
67     while(encoders_count < tickGoal){
68         encoders_count = (encoders.getCountsLeft()+encoders.getCountsRight())/2;
69         calculateNewRobotCordinate();
70         motors.setSpeeds(V_MAX, V_MAX);
71     }
72 }
73
74 void distanceCheckHandler(){
75     calculateDistanceToGoal();
76
77     reachDistance();
78
79     // Serial.println(distance);
80     if(currentGoalDot < GOAL_DOTS_NUMBER){
81         calculateGoalAngle(pathYCoordinates[currentGoalDot], pathXCoordinates[currentGoalDot]);
82         currentGoalDot++;
83         rightDirection=false;
84     }else{
85         //Uncomment next two lines for repeating of following dots
86         calculateGoalAngle(pathYCoordinates[0], pathXCoordinates[0]);
87         currentGoalDot = 1;
88
89         //Comment line bellow for infinity dots follow
90         // goalReached = true;
91     }
92 }
```

- Izračunamo udaljenost od trenutne pozicije robota do željene točke,
- Primijenimo izračunatu udaljenost kako bi odredili broj tickova enkodera koje mora prevaliti kako bi prešao željenu udaljenost,
- Pokrenimo robota prema naprijed,
- Ažurirajmo nove koordinate robota



## 4.5 Crtanje kvadra s robotom

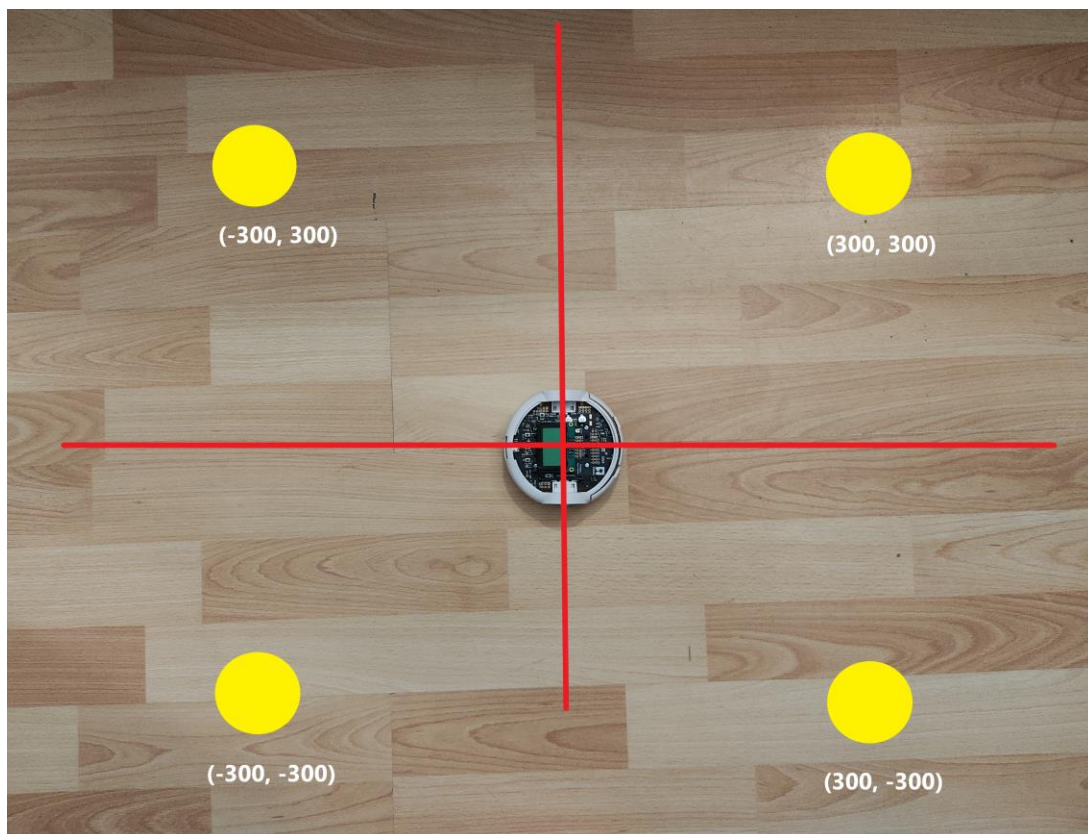
Početna točka robota je (0,0) s kutom 0 stupnjeva u odnosu na x os. Dok su ciljane točke koje robot treba „pronaći“ redom:

1. (-300, 300),
2. (-300, -300),
3. (300, -300),
4. (300, 300)

Zapisane u polje

```
23 //Square dots
24 int pathYCoordinates [GOAL_DOTS_NUMBER] = {300, -300, -300, 300};
25 int pathXCoordinates [GOAL_DOTS_NUMBER] = {-300, -300, 300, 300};
26
```

NAPOMENA: Robot se uvijek ne okreće najoptimalnije za dolazak u slijedeću točku jer za računanje kuta koristi atan2 funkciju koja ograničava kut na (-pi, pi).



## 5 Zaključak

Enkoderi su korisna stavka svakog robota, ali nisu dovoljno pouzdani sami za sebe ukoliko se koriste za lociranje robota u prostoru. Već skoro svaki robot koji pronalazi nekakvu primjenu opremljen je žiroskopima, akcelerometrima.

Potpomognuti jedni drugima daju preciznu i točnu lokaciju robota u prostoru.