

**UNIVERZA V LJUBLJANI**

Fakulteta za strojništvo

**Razvojno okolje za vizualno programiranje  
mobilnih robotov**

Diplomsko delo Visokošolskega strokovnega študijskega programa I. stopnje  
**STROJNIŠTVO**

**Dušan Majkić**

Ljubljana, junij 2017



**UNIVERZA V LJUBLJANI**

Fakulteta za strojništvo

**Razvojno okolje za vizualno programiranje  
mobilnih robotov**

Diplomsko delo Visokošolskega strokovnega študijskega programa I. stopnje  
**STROJNIŠTVO**

**Dušan Majkić**

Mentor: doc. dr. Rok Vrabič, univ. dipl. inž.  
Somentor: prof. dr. Peter Butala, univ. dipl. inž.

Ljubljana, junij 2017



Kandidat  
Dušan MAJKIĆ

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM I. STOPNJE STROJNITVO: VS I/412

NASLOV TEME: Razvojno okolje za vizualno programiranje mobilnih robotov

Fakulteta za strojništvo, UL v sklopu poletne šole nudi delavnico »Mobilni robot«, na kateri osnovno- in srednješolci sestavljajo in programirajo mobilne robote. Pri tem se mnogi udeleženci prvič srečajo s programiranjem. V zadnjih letih se pri poučevanju programiranja v zgodnji starosti vse bolj uveljavljajo vizualni programski jeziki, ki pisanje kode nadomestijo z zlaganjem in povezovanjem grafičnih blokov. Cilj diplomskega dela je pregledati, izbrati in dograditi obstoječo razvojno okolje za vizualno programiranje z zmožnostjo programiranja robotov, ki temeljijo na mikrokrumilniški platformi Arduino.

V diplomskem delu preglejte stanje tehnike na področju vizualnega programiranja in pripadajočih razvojnih okolij. Izberite ustrezno okolje glede na možnosti nadgrajevanja za programiranje mikrokrumilnikov. Preglejte zgradbo in funkcionalnosti obstoječega mobilnega robota ter določite nabor in obliko funkcij, s katerimi je možno celostno obvladovanje robotovih senzorjev in aktuatorjev. Nadgradite izbrano okolje s tem naborom funkcij. Pri tem določite tudi njihovo grafično podobo. Predstavite postopek povezave vizualnega programa in izvodne kode za robotov mikrokrumilnik. Uspešnost nadgradnje dokažite z demonstracijami na realnem robotu.

Diplomsko delo je treba oddati v jezikovno in terminološko pravilni slovenščini.

Rok za oddajo tega dela je šest mesecev od dneva prevzema.

Mentor

doc. dr. Rok Vrabič, univ. dipl. inž.

Somentor

prof. dr. Peter Butala, univ. dipl. inž.

Predsednik diplomske komisije

prof. dr. Janez Tusek, univ. dipl. inž.

Podpisani sem delo

izvedovalca/abonent

20.04.2017



Prodekan za pedagoško dejavnost I. stopnje  
prof. dr. Roman Šturm, univ. dipl. inž.



## **Zahvala**

---

Zahvala gre v prvi vrsti mentorju, doc. dr. Roku Vrabiču, in somentorju prof. dr. Peter Butala, za njuno svetovanje, trud in pomoč pri izdelavi diplomske naloge. Zahvaljujem se tudi svojim staršem, sošolcem in prijateljem za njihovo moralno podporo in pomoč, ter vsem ostalim, ki so mi na kakršen koli način pomagali do uspešnega zaključka študija.



## Izjava

---

Spodaj podpisani Dušan Majkić študent Fakultete za strojništvo Univerze v Ljubljani, z vpisno številko 23120742, avtor pisnega zaključnega dela študija z naslovom: Razvojno okolje za vizualno programiranje mobilnih robotov,

IZJAVLJAM,

1. a) da je pisno zaključno delo študija rezultat mojega samostojnega dela;
- b) da je pisno zaključno delo študija rezultat lastnega dela več kandidatov in izpolnjuje pogoje, ki jih Statut UL določa za skupna zaključna dela študija ter je v zahtevanem deležu rezultat mojega samostojnega dela;
2. da je tiskana oblika pisnega zaključnega dela študija istovetna elektronski obliki pisnega zaključnega dela študija;
3. da sem pridobil/-a vsa potrebna dovoljenja za uporabo podatkov in avtorskih del vpisnem zaključnem delu študija in jih v pisnem zaključnem delu študija jasno označil/-a;
4. da sem pri pripravi pisnega zaključnega dela študija ravnal/-a v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil/-a soglasje etične komisije;
5. da soglašam z uporabo elektronske oblike pisnega zaključnega dela študija za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;
6. da na UL neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve avtorskega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja pisnega zaključnega dela študija na voljo javnosti na svetovnem spletu preko Repozitorija UL;
7. da dovoljujem objavo svojih osebnih podatkov, ki so navedeni v pisnem zaključnem delu študija in tej izjavi, skupaj z objavo pisnega zaključnega dela študija;
8. da dovoljujem uporabo mojega rojstnega datuma v zapisu COBISS.

V Ljubljani, \_\_\_\_\_

Podpis avtorja/-ice: \_\_\_\_\_



# Izvleček

---

UDK 004.4:007.52(043.2)

Tek. štev.: VS I/412

## Razvojno okolje za vizualno programiranje mobilnih robotov

Dušan Majkić

Ključne besede: izobraževanje  
vizualno programiranje  
mobilni roboti  
Arduino  
Ardublockly

Na Fakulteti za strojništvo Univerze v Ljubljani poteka vsako leto poletna šola, v sklopu katere se izvaja delavnica »Mobilni robot«. Na delavnici osnovno- in srednješolci spoznavajo delovanje mobilnega robota, kar zajema tudi njegovo programiranje, pri čemer se nekateri otroci prvič srečajo s programiranjem. Z namenom približevanja programiranja otrokom so se v zadnjem desetletju razvila programska okolja oz. programski jeziki, ki za uspešno delo s programskim jezikom od uporabnika ne zahtevajo nobenega predznanja, saj programiranje temelji na uporabi grafičnih objektov. Cilj diplomske naloge je bila izbira in izvedba modifikacije izbranega vizualnega programskega okolja z namenom programiranja mobilnih robotov. Modifikacija programskega okolja je potekala preko dodajanja vrstic kode v izvorno kodo programa. Končni rezultat je bil modificirano vizualno programsko okolje z vsebovanimi novimi bloki kode, ki omogočajo programiranje mobilnega robota. Za konec je bil z namenom testiranja predelave in z uporabo naših novih blokov kode sprogramiran enostaven program gibanja mobilnega robota.



# **Abstract**

---

UDC 004.4:007.52(043.2)

No.: VS I/412

## **Development environment for visual programming of mobile robots**

Dušan Majkić

Key words: education  
visual programming  
mobile robots  
Arduino  
Ardublockly

At Faculty of Mechanical Engineering of the University of Ljubljana a summer school takes place every year, within which a workshop »Mobile robot« is organized. At the workshop pupils from elementary and high schools learn about mobile robots and their programming. At this age most of the pupils do not have any knowledge about programming. Therefore, in the past decade new programming languages were developed. These programming languages do not require any special knowledge about programming, because the coding is based on the use of graphical objects. The aim of this diploma thesis was to choose a visual integrated development environment, which supports the Arduino platform, and to modify the chosen visual integrated development environment to support robot programming. Modification of the program was made by inserting new lines of code in the multiple folders of the program's source code. The end result was modified visual integrated development environment with included blocks of code to program the mobile robot. In order to make a test of the modification the simple a robot moving program was made with the use of the new blocks.



# Kazalo

---

<b>Zahvala .....</b>	<b>vi</b>
<b>Izjava.....</b>	<b>viii</b>
<b>Izvleček .....</b>	<b>x</b>
<b>Abstract .....</b>	<b>xii</b>
<b>Kazalo .....</b>	<b>xiv</b>
<b>Kazalo slik .....</b>	<b>xvi</b>
<b>Seznam uporabljenih simbolov .....</b>	<b>xviii</b>
<b>Seznam uporabljenih okrajšav .....</b>	<b>xx</b>
 <b>1. Uvod.....</b>	 <b>1</b>
1.1.    Cilji.....	2
 <b>2. Izobraževanje in robotika .....</b>	 <b>3</b>
2.1.    Pregled robotskih kompletov.....	3
2.1.1. Lego MINDSTORMS .....	3
2.1.2. Cubelets .....	5
2.1.3. Atoms.....	6
2.1.4. Fischertechnik.....	7
2.1.5. ROKIT smart .....	8
2.1.6. Dash in Dot.....	9
2.2.    Robotika v izobraževalnem procesu .....	10
 <b>3. Vizualno programiranje .....</b>	 <b>13</b>
3.1.    Razvoj programiranja.....	13
3.2.    Predstavitev vizualnega programiranja .....	13
3.3.    Področja uporabe .....	15
3.4.    Delitev vizualnih programskeh jezikov .....	16
3.5.    Prednosti in pomanjkljivosti .....	18
 <b>4. Pregled stanja in izbor vizualnega okolja.....</b>	 <b>19</b>
4.1.    NETLab Toolkit.....	19

4.2.	<b>Ardublock.....</b>	<b>20</b>
4.3.	<b>S4A Scratch .....</b>	<b>21</b>
4.4.	<b>Modkit .....</b>	<b>21</b>
4.5.	<b>MiniBloq .....</b>	<b>22</b>
4.6.	<b>Visuino .....</b>	<b>23</b>
4.7.	<b>Blockly .....</b>	<b>24</b>
4.8.	<b>Ardublockly.....</b>	<b>24</b>
4.9.	<b>Izbor vizualnega okolja .....</b>	<b>25</b>
<b>5.</b>	<b>Opis mobilnega robota.....</b>	<b>27</b>
5.1.	<b>Zgradba mobilnega robota .....</b>	<b>27</b>
5.2.	<b>Funkcije mobilnega robota .....</b>	<b>29</b>
<b>6.</b>	<b>Nadgradnja vizualnega okolja .....</b>	<b>31</b>
6.1.	<b>Modifikacija liste orodij (Toolbox) .....</b>	<b>31</b>
6.2.	<b>Ustvarjanje novega bloka .....</b>	<b>33</b>
6.2.1.	Definicija bloka .....	33
6.2.2.	Generiranje kode bloka .....	34
6.2.3.	Umetitev kode .....	34
6.2.4.	Blok Pelji .....	37
6.3.	<b>Rezultati modifikacije in opis novih funkcijskih blokov.....</b>	<b>39</b>
<b>7.</b>	<b>Demonstracija na realnem robotu.....</b>	<b>41</b>
<b>8.</b>	<b>Zaključek.....</b>	<b>45</b>
<b>9.</b>	<b>Literatura .....</b>	<b>47</b>
<b>10.</b>	<b>Priloga 1 – Knjižnica Robot.h .....</b>	<b>51</b>
<b>11.</b>	<b>Priloga 2 – knjižnica Robot.cpp .....</b>	<b>53</b>

# Kazalo slik

---

Slika 1: Modul EV3 s pripadajočimi tipali [1].....	4
Slika 2:Izgled grafičnega virtualnega okolja LEGO MINDSTORMS [1].....	5
Slika 3: Izgled robota Cubelets [3].....	5
Slika 4: Izgled programskega okolja Cubelets Blockly Atoms kit [4].....	6
Slika 5: Izgled kompleta Atoms [6]. .....	7
Slika 6: Izgled mobilnega robota Fischertechnik in pripadajoča programska oprema [8].....	8
Slika 7: Izgled vizualnega programskega okolja Robo pro [9].....	8
Slika 8: Roboti iz učnega kompleta ROKIT Smart [11]. .....	9
Slika 9: Izgled robotov Dash and Dot tablice z aplikacijo Blockly [13].....	10
Slika 10: Primer tekstovne kode v programskem jeziku C++ [20]. .....	14
Slika 11: Primer blokovnega vizualnega programiranja [21]. .....	14
Slika 12: Prikaz programiranja z uporabo ikon [24]. .....	16
Slika 13: Prikaz programiranja z uporabo obrazcev (Forms/3) [25].....	17
Slika 14: Prikaz programiranja z uporabo diagramov [27]. .....	17
Slika 15: Izgled vizualnega programskega okolja NTK [32].....	20
Slika 16: Izgled vizualnega programskega okolja Ardublock [33].....	20
Slika 17: Izgled vizualnega okolja S4A Scratch [34].....	21
Slika 18: Izgled vizualnega okolja Modkit [35].....	22
Slika 19: Izgled vizualnega okolja MiniBloq [36]. .....	23
Slika 20: Izgled vizualnega okolja Visuino [37].....	23
Slika 21: Izgled vizualnega okolja Blockly [38].....	24
Slika 22: Izgled vizualnega okolja Ardublockly [39]. .....	25
Slika 23: Mobilni robot zgornji del (levo) in spodnji del (desno) [41]. .....	27
Slika 24: Prikaz vezalne sheme mobilnega robota [41]. .....	28
Slika 25: Izgled JavaScript programske kode za funkcijo attachinterrupt. .....	29
Slika 26: Izgled komprimirane kode. .....	31
Slika 27: Izgled dekomprimirne kode. .....	32
Slika 28: Dodatek xml kode za razširitev obstoječe liste orodij. .....	32
Slika 29: Dodana kategorija kot rezultat spremembe xml kode.....	33
Slika 30: Prikaz zavihka Blockly Factory [40]. .....	34
Slika 31: Generirana koda za izvoz testnega bloka [40]. .....	35
Slika 32: Vstavljen script stavek z lokacijo datoteke s kodo definicije bloka.....	35
Slika 33: Vstavljen script stavek z lokacijo datoteke z generirajoče kodo.....	36
Slika 34:Vstavljenha vrstica za naš testni blok. .....	36
Slika 35: Prikaz novo ustvarjenega bloka v programu Ardublockly.....	36
Slika 36: Generirajoča koda funkcije pelji.....	37
Slika 37: Prikaz kode definicije bloka pelji.....	37
Slika 38: Sprememba imena bloka.....	38

Slika 39: Prikaz lokacije navezave na datoteko definicije bloka pelji.	38
Slika 40: Prikaz lokacije navezave na datoteko generirajoče kode.	38
Slika 41: Prikaz spremembe našega testnega bloka v blok pelji.	39
Slika 42: Končni izgled našega programskega okolja z vstavljenimi novimi bloki.	39
Slika 43: Prikaz prvega testnega programa.	41
Slika 44: Prikaz drugega testnega programa.	42
Slika 45: Poenostavljen blokovni diagram mobilnega robota.	43
Slika 46: Mobilni robot na testni stezi z novo ustvarjenim programom.	43

## Seznam uporabljenih simbolov

---

Oznaka	Enota	Pomen
$K$	/	koeficient ojačanja
$I$	/	izhodni signal
$O$	/	odstopek
$R$	/	referenčni signal
$T$	s	čas
$V$	m/s	premočrtna hitrost
$\mathcal{Q}$	1/s	kotna hitrost
<hr/>		
Indeksi		
<hr/>		
P		proporcionalno



# Seznam uporabljenih okrajšav

---

Okrajšava	Pomen
FERI	Fakulteta za elektrotehniko, računalništvo in informatiko Univerze v Mariboru
FLL	liga robotskega LEGO tekmovanja (ang. <i>First LEGO league</i> )
IDE	integrirano razvojno okolje (ang. <i>Integrated Development Environment</i> )
ISR	prekinitvena rutina (ang. <i>Interrupt Service Routine</i> )
IR	infrardeče valovanje (and. <i>Infrared</i> )
LAKOS	laboratorij za tehnično kibernetiko, obdelovalne sisteme in računalniško tehnologijo
LED	svetleča dioda (ang. <i>Light-Emitting Diode</i> )
PC	osebni računalnik (ang. <i>Personal Computer</i> )
S4A	Scratch za Arduino (ang. <i>Scratch for Arduino</i> )
USB	Univerzalno serijsko vodilo (ang. <i>Universal Serial Bus</i> )
VPL	vizualni programski jezik (ang. <i>Visual Programming Language</i> )
VPE	vizualno programsko okolje (ang. <i>Visual Programming Environment</i> )
3D	trirazsežni prostor (ang. <i>Three-dimensional space</i> )
2D	dvorazsežni prostor (ang. <i>Two-dimensional space</i> )



# 1. Uvod

Živimo v času, v katerem smo priča skoraj vsakodnevnu pojavu novih izdelkov na trgu in uvajanju ter uporabi novih tehnologij. Tako se v industriji pojavlja uporaba novih izdelovalnih tehnologij, naprednejša programska oprema, večji nadzor nad podatki, uporabljajo se vzdržljivejša orodja, novi napredni materiali in povečuje se stopnja avtomatizacije.

Na razmah avtomatizacije je v zadnjih desetletjih v večji meri vplival razvoj na področju elektrotehnike, saj so avtome včasih predstavljeni stroji, ki so izključno temeljili na mehanskem principu, sedaj pa mehanske komponente nadomeščajo, mnogo manjše, elektronske komponente. Zato je sedaj mogoče izdelati mnogo manjše avtomatske naprave za različne namene uporabe, ki so bili včasih nepoznani in takratnemu človeku nepredstavljeni. Stopnja avtomatizacije je v industrijski panogi povezana tudi z robotiko oz. s številom uporabljenih robotov v procesu proizvodnje. Ne smemo zanemariti dejstva, da se vzporedno s stopnjo avtomatizacije v industriji povečuje tudi stopnja avtomatizacije med končnimi uporabniki. Tako imamo na trgu vse več naprav, ki delujejo na principu senzorjev in mikrokrmlnikov. V to skupino štejemo moderne kavne avtome, igralne avtome, bankomate, termostate, avtomobilske sisteme za pomoč pri vožnji, elektronske table ... Mogoče je opaziti trend vse večjega pojava in uporabe avtomatskih naprav, zato lahko v prihodnosti pričakujemo vedno večje število avtomatiziranih procesov na skoraj vseh področjih našega življenja.

Razvoju tehnologij posledično sledi tudi izobraževalni proces, v katerem se povečuje uporaba računalnikov, programov in drugih naprednih digitalnih pripomočkov, s pomočjo katerih se otroci učijo in usvajajo nove tehnologije. Tako list papirja in svinčnik v določenih primerih nadomešča računalnik ali tablica, kocke pa nadomeščajo napredne igrače. Zato lahko trdimo, da je način podajanja snovi s pomočjo table nadomestil nov koncept podajanja snovi, in sicer preko uporabe naprednih tehnologij. Ker je okolica otrok, v kateri se nahajajo, vse bolj modernizirana, otroci vedno pogosteje in v zgodnejših letih prihajajo v stik z novimi tehnologijami oz. v našem primeru z robotiko. Z namenom približevanja in poučevanja robotike otrok kot tudi širše populacije so bili razviti različni učni robotske kompleti. Kompleti so prilagojeni starosti uporabnika, zato je mogoče zaslediti manj zahtevne kompleti, ki so pripravljeni za takojšnjo uporabo, in kompleti za izkušenejše, ki so bolj prilagodljivi in uporabni, vendar zahtevajo od uporabnika nekatera predznanja. Na ta način se otroci na zabaven in zanimiv način srečujejo z robotiko in se učijo programiranja robota. Drug način prenosa znanja je sodelovanje otrok na poletnih in počitniških šolah ter

delavnicah ali pa sodelovanje v različnih tekmovanjih na državni kot tudi na mednarodni ravni.

Tako tudi Fakulteta za strojništvo Univerze v Ljubljani v sklopu Poletne šole strojništva nudi delavnico »Mobilni robot«, na kateri osnovno- in srednješolci sestavljajo in programirajo mobilne robote. Pri tem se mnogi udeleženci prvič srečajo s programiranjem. V zadnjih letih se pri poučevanju programiranja v zgodnji starosti vse bolj uveljavljajo vizualni programski jeziki, ki pisanje kode nadomeščajo z zlaganjem in povezovanjem grafičnih blokov v program. Na ta način je učencem olajšan postopek in omogočeno hitrejše dojemanje principa programiranja.

V nadaljevanju uvoda bodo najprej definirani in opredeljeni cilji, katerih zaporedje bo vodilo za razvoj vsebine od začetka in vse do konca diplomske naloge. Uvodnemu poglavju sledi poglavje pregleda in vključevanja robotike v izobraževalne procese, pri čemer bomo posvetili večjo pozornost neformalnemu učenju robotike. Omenjena oblika izobraževanja obsega predstavitev robotskega tekmovanja, tako v slovenskem kot v mednarodnem prostoru, ter predstavitev poletnih šol in delavnic. V istem poglavju bodo opisani tudi najbolj poznani in najpogosteje uporabljeni robotski učni kompleti, katerih uporaba sega tudi na področje robotskih tekmovanj. V poglavju vizualnega programiranja bo uvodoma podana kratka zgodovina razvoja vizualnega programiranja, in sicer kako in zakaj je prišlo do nastanka tovrstnega jezika, čemur sledi kratka teoretična predstavitev in delitev vizualnih programskih jezikov. V istem poglavju sledi pregled izbranih vizualnih okolij in navedba, v kakšni obliki se v posameznem vizualnem okolju uporablja programski vizualni jezik. Pregled izbranih vizualnih okolij nam bo služil kot osnova za izvedbo selekcije vizualnega okolja, na katerem bomo kasneje izvedli modifikacijo. Opravljenemu izboru bo sledil opis mobilnega robota in vse funkcije, ki jih bomo vgradili v izbrano programsko okolje. Za eno od novo vgrajenih funkcij bo detajlno opisan postopek implementacije funkcije v programsko okolje. Za testiranje uspešnosti modifikacije bomo program, ki ga bomo zgradili iz novih blokov v izbranem programskem okolju, preizkusili s pomočjo mobilnega robota na testni stezi. V zaključku sledi analiza in ugotovitve opravljenega dela.

## 1.1. Cilji

Cilji diplomske naloge so:

- pregledati stanje robotike za izobraževalne namene,
- pregledati stanje na področju vizualnih programskih okolij, ki temeljijo na mikrokrmilniški platformi Arduino in imajo možnost nadgradnje,
- izbrati in nadgraditi izbrano vizualno programsko okolje,
- s pomočjo modificiranega programa napisati program gibanja za avtonomno gibanje mobilnega robota in
- testirati ustvarjeno kodo

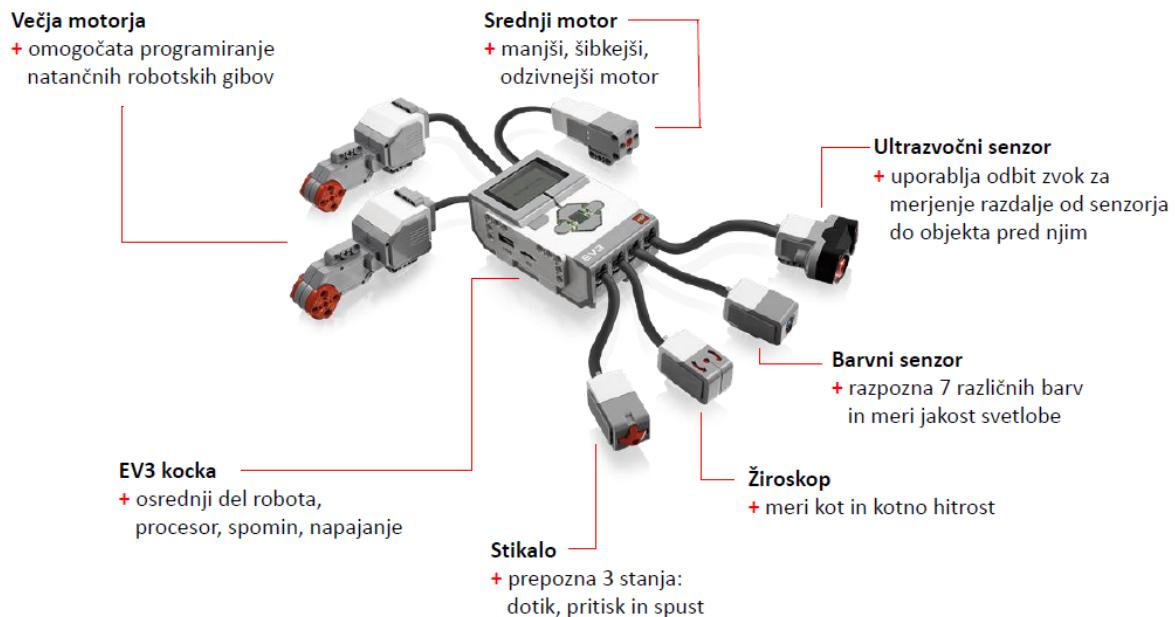
## **2. Izobraževanje in robotika**

Da bi približali robotiko in z njo povezano programiranje širši populaciji, so se na trgu pojavili raznovrstni učni kompleti programljivih igrač s pripadajočo programsko opremo. V večji meri so kompleti namenjeni otrokom in mladostnikom, obstajajo pa tudi nekateri zahtevnejši kompleti za izobraževanje na višji ravni in za vse ostale privržence robotike. Uporaba nekaterih kompletov v izobraževalne namene je s časom prerasla v organizirana tekmovanja. Zato bodo v nadaljevanju poglavja predstavljeni nekateri robotski kompleti, namenjeni izobraževanju oz. igri, in predstavljena nekatera slovenska in mednarodna mladinska robotska tekmovanja

### **2.1. Pregled robotskih kompletov**

#### **2.1.1. Lego MINDSTORMS**

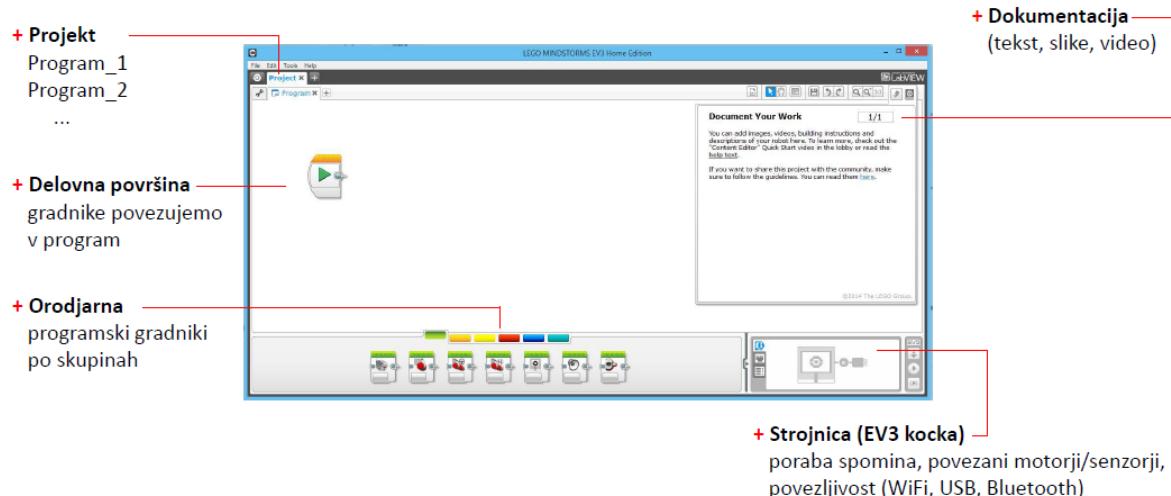
Učni komplet MINDSTORMS je izdelek oz. igrača podjetja LEGO, za otroke ali navdušence nad robotiko, pri čemer s pomočjo uporabe plastičnih kock gradimo najrazličnejše projekte. Izraz MINDSTORMS bi lahko prevedli kot "viharjenje uma", kar pomeni spodbujanje ustvarjalnega mišljenja. S kompletom MINDSTORMS lahko sestavimo robota po svojih željah in potrebah, kar omogoča uporabniku uporabo kompleta na različnih področjih. Komplet lahko uporabljam v izobraževalne namene, kar vzpodbuja uporabnika k logičnem mišljenju in uresničevanju njegovih zamisli. Programsко okolje omogoča dodatno nadgrajevanje s strani uporabnikov, podjetja LEGO in partnerskih podjetij.



Slika 1: Modul EV3 s pripadajočimi tipali [1].

Oznaka EV3 označuje tretjo generacijo učnega kompleta MINDSTORMS. Z isto oznako EV3 označujemo tudi glavno kocko, ki predstavlja krmilniško enoto robota. Servomotorji omogočajo premike robota ali njegovih delov, za zaznavanje okolice pa je na voljo več tipal (ultrazvočno, svetlobno, zvočno itd.). Modul EV3 lahko programiramo na osebnem računalniku, ki ga povežemo prek Bluetooth ali USB-ja. Tehnologija Bluetooth nam omogoča povezavo robota na daljavo z uporabo dlančnika ali mobilnega telefona.

LEGO MINDSTORMS robote lahko programiramo z različnimi programskimi jeziki, kot so: C, Matlab, LabWiew, Java ... Programiramo lahko tudi v posebnem programskem okolju, ki ga lahko uporabljajo tudi ljudje brez predznanja programiranja, saj programsko okolje ne zahteva pisanja nobene programske kode. Programiranje poteka z zlaganjem grafičnih gradnikov, ki so v orodjarni razdeljeni v posamezne skupine glede na njihovo funkcijo v programu. Orodjarna je sestavljena iz petih skupin, in sicer iz skupine akcij, poteke, senzorjev, podatkov in skupine z naprednimi gradniki. Po potrebi lahko uporabnik ustvari svojo skupino, v katero lahko vstavlja nove gradnike za svoje potrebe [1], [2].



Slika 2: Izgled grafičnega virtualnega okolja LEGO MINDSTORMS [1].

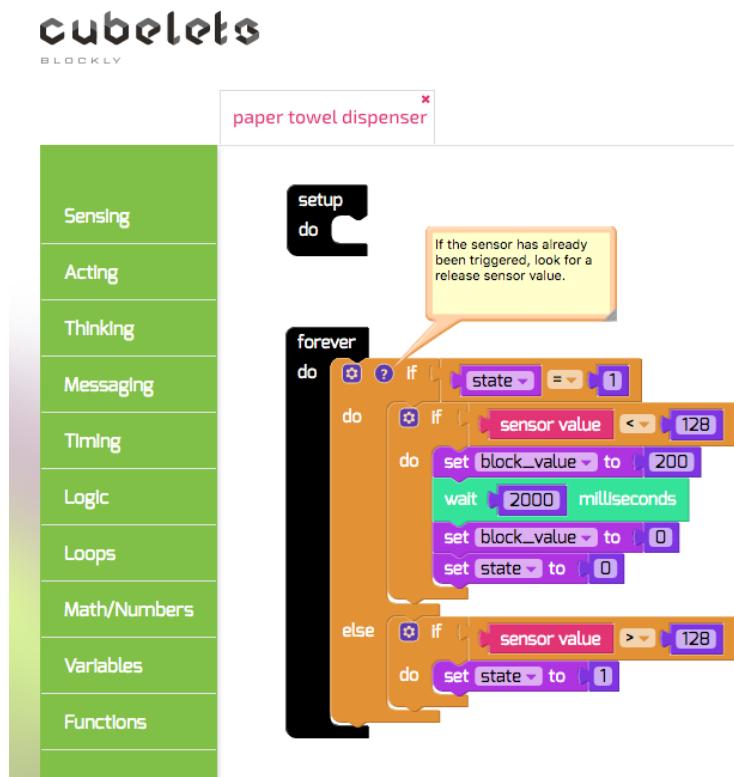
### 2.1.2. Cubelets

Preko sestavljanja treh tipov robotskih kock Cubelets lahko ustvarimo robota. Glede na postavitev posameznih kock, opravlja robot določeno funkcijo, pri čemer lahko funkcijo spremenimo z menjavo ali dodajanjem drugih kock. Ločimo tri tipe kock, in sicer zaznavalne, izvršilne ter logične kocke, pri čemer so zaznavalne kocke bele barve, izvršile kocke prosojne barve in logične kocke rjave barve.



Slika 3: Izgled robota Cubelets [3].

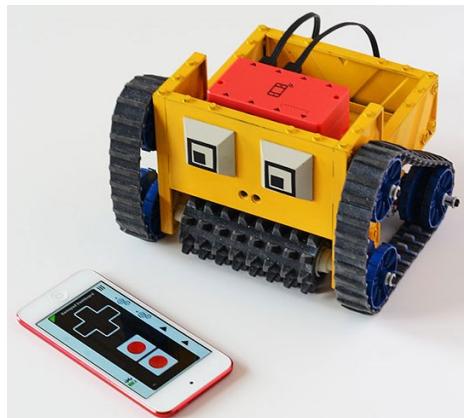
Večina kock ima pet povezovalnih ploskev in eno, po kateri ločujemo posamezne kocke med seboj. Ostale kocke imajo šest povezovalnih ploskev, pri čemer njihovo funkcijo prepoznamo glede na njihovo barvo. Vsaka kocka ima v kotu majhno LED lučko. LED lučka se prižge v trenutku, ko robotu dodamo napajalno kocko. Za vsak sestavljen robot potrebujemo najmanj tri koce, in sicer zaznavalno, izvršno ter napajalno kocko. Kocke Cubelets lahko programiramo preko uporabe modre Bluetooth kocke, ki predstavlja povezavo med računalnikom in robotom. Programiramo lahko s pomočjo programa Cubelets Flash v programskega jeziku C ali s pomočjo grafičnega programskega okolja Cubelets Blockly [3], [4].



Slika 4: Izgled programskega okolja Cubelets Blockly Atoms kit [4].

### 2.1.3. Atoms

Atoms je sistem, ki temelji na uporabi pametnih kock in je zasnovan za otroke. Kompleti omogočajo gradnjo in dodajanje interaktivnih funkcij posameznim stvaritvam. Obstaja več tematskih sklopov kompletov, ki se lahko uporablajo v kombinaciji z Lego kockami, lahko so všite v tkanino ali pribite v les. Atoms kompleti so raznoliki, vzdržljivi in cenovno ugodni ter zasnovani tako, da lahko kateri koli otrok razume delovanje igrače v petih minutah igre. Na ta način je uporabniku podana tehnologija na razumljiv, enostaven in zabaven način. Pametne kocke razvrščamo glede na njihove funkcije v štiri skupine, in sicer v skupino zaznaval, napajanja, logike in izvršitve. Vse omenjene kocke iz posameznih skupin lahko med seboj povezujemo v različne kombinacije. Preko Bluetooth povezave in posebne aplikacije za nadzor Atoms pametnih kock, jih lahko nadziramo in upravljamo preko mobilnega telefona [5], [6].



Slika 5: Izgled kompleta Atoms [6].

#### 2.1.4. Fischertechnik

Konstrukcijski kompleti Fischertechnik omogočajo sestavljanje različnih modelov, ki realistično prikazujejo delovanje pravih strojev in naprav. Z uporabo različnih gradnikov, motorjev, senzorjev, luči, programske opreme in računalniškega nadzora se uporabnikom odpirajo povsem nove možnosti raziskovanja, načrtovanja in preizkušanja skonstruiranih naprav. Kompleti so zasnovani tako, da spodbujajo logično razmišljanje in pridobivanje osnovnega tehničnega znanja na neformalen in nezahteven način. Gradniki iz različnih kompletov so med seboj združljivi, zato jih lahko med seboj poljubno kombiniramo. Tako se z uporabo gradnikov iz različnih kompletov odpira veliko novih konstrukcijskih možnosti.

Osnovo vseh konstrukcijskih kompletov Fischertechnik predstavlja osnovni gradnik Fischertechnik, ki ga lahko spajamo na vseh 6 stranicah. Konstrukcijski kompleti so razvrščeni v štiri stopnje zahtevnosti, ki se med seboj dopolnjujejo in nadgrajujejo. Vsak izmed kompletov omogoča stik z resnično tehnologijo in njeno spoznavanje med igro. Glede na starostne skupine in tehnično predznanje ločimo komplete Junior, Basic + Advanced, Profi in Computing.

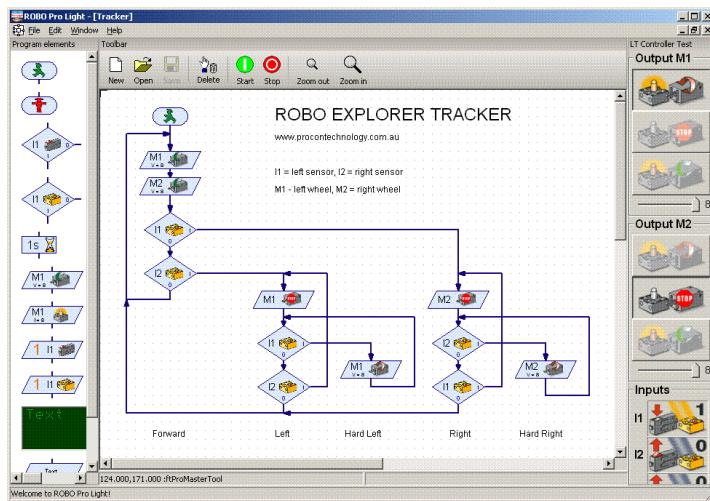
Izobraževalni konstrukcijski kompleti Fischertechnik iz programa PROFI so posebej zasnovani za uporabo pri tehnični vzgoji in kot didaktični pripomoček pri naravoslovnih predmetih. Kompleti se uporabljam pri spoznavanju mehanike, statike, elektrotehnike, mehatronike (merjenje, krmiljenje, regulacija) in ekologije (pridobivanje čiste energije). V dodatno pomoč so na voljo tudi delovni priročniki in delovni listi. Na voljo so tudi študijski modeli, ki predstavljajo kompaktne delujoče modele in so ob dobavi že sestavljeni. Študijski modeli tako predstavljajo izvrsten študijski in demonstracijski pripomoček za izobraževanje, izpopolnjevanje in razvoj industrijske avtomatizacije.

Za nas je zanimiva skupina kompletov Computing, ki vsebuje komplet mobilnega robota. Komplet vsebuje 480 sestavnih delov, iz katerih lahko uporabniki sestavijo različne modele mobilnih robotov. S pomočjo programljivega vmesnika ROBO INTERFACE in s programsko opremo ROBO PRO lahko upravljamo in programiramo delovanje robota. Robot vsebuje senzor svetlobe in senzor črte ter lahko zaznava ovire. Komplet vsebuje delovni priročnik "Programiranje in krmiljenje robotov Fischertechnik s PC računalnikom" (Programming and Control of Fischertechnik Robots with the PC) [7].



Slika 6: Izgled mobilnega robota Fischertechnik in pripadajoča programska oprema [8].

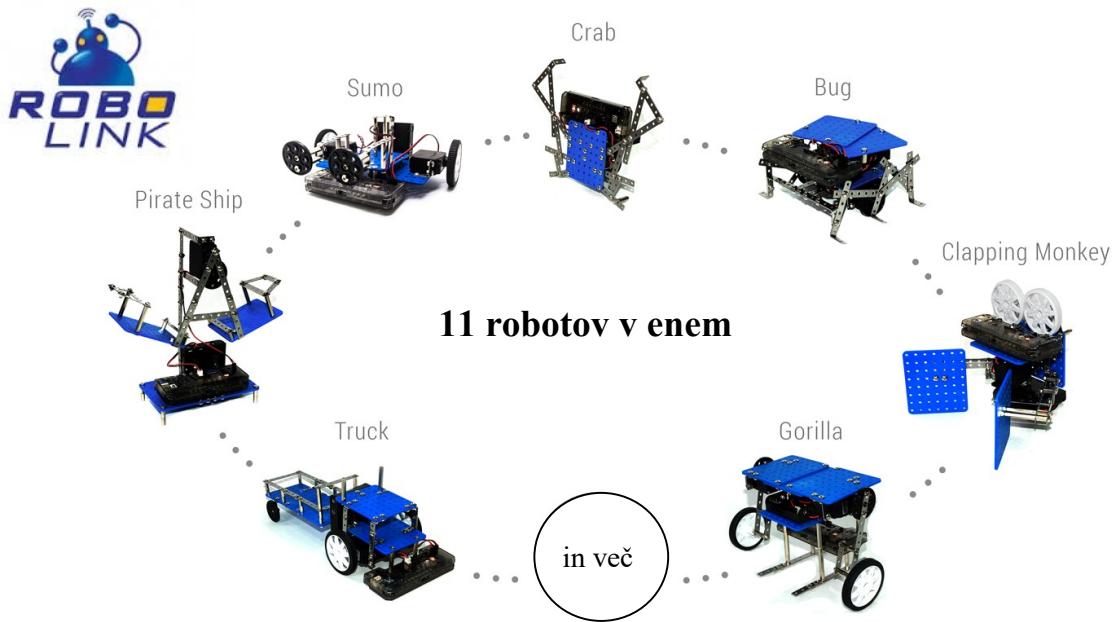
Programiranje mobilnega robota temelji na uporabi grafičnega programskega okolja Robo pro, v katerem s sestavljanjem blokov tvorimo program. Celoten program ima obliko diagrama, ki prikazuje medsebojne povezave med posameznimi bloki [7].



Slika 7: Izgled vizualnega programskega okolja Robo pro [9].

### 2.1.5. ROKIT smart

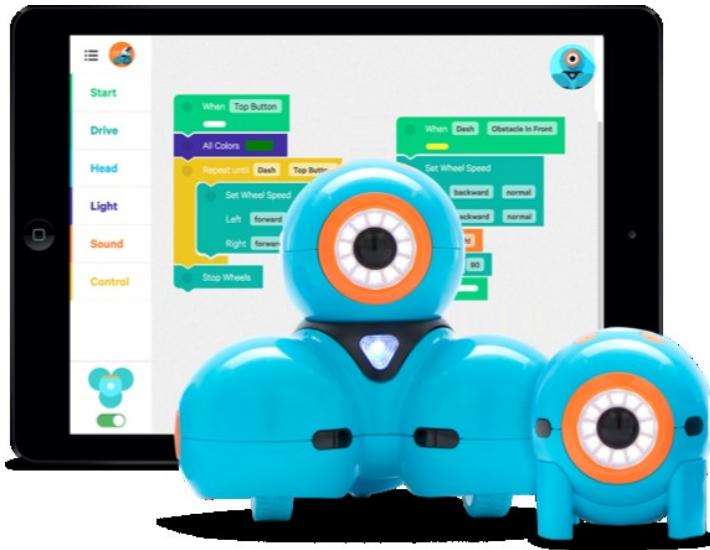
Rokit Smart je uporabniku prijazen robotski komplet, v katerem se nahaja več standardnih komponent iz sveta robotike. Komplet je namenjen predvsem otrokom, ki se preko uporabe oz. igre s kompletom učijo gradnje in programiranja robotov. Glede na priložene komponente v kompletu je mogoče z njihovo uporabo v različnih kombinacijah sestaviti do 11 različnih robotov. Glavni del kompleta predstavlja Rokit »pametna« ploščica oz. krmilnik. Ploščica vsebuje tri IR senzorje z namenom meritve razdalj, tako sta dva senzorja nameščena na vsaki strani robota in eden na sredini. Dodatno se na spodnji strani ploščice nahaja sedem IR senzorjev. Povezavo z zunanjimi zaznavalci je mogoče izvesti preko uporabe petih analognih ali petih digitalnih pinov. Na ploščico lahko priključimo največ 4 motorje, in sicer preko uporabe štirih pinov. Ker je mogoče robota tudi programirati, se za prenos programske kode uporablja USB priključek. Robota je mogoče upravljati tudi na razdaljo preko uporabe daljinca [10].



Slika 8: Roboti iz učnega kompletta ROKIT Smart [11].

### 2.1.6. Dash in Dot

Podjetje Wonder workshop je ustvarilo robota z imenoma Dash in Dot. Dash in Dot sta prava robota, namenjena otrokom, ki se tako učijo programiranja skozi igro. Z uporabo posebno prilagojenih aplikacij na tablici ali pametnem telefonu se otroci učijo programiranja tako, da ukažejo robotu, da poje, pleše ali se giblje po prostoru. Robota imata vgrajene senzorje z namenom zaznavanja okolice in preprečevanja nepričakovanih trkov. Za programiranje robotov je na voljo več aplikacij, preko katerih se otroci srečajo s programiranjem. Na voljo so aplikacije Wonder, Blockly, Xylo, Path in Go. Aplikacija Wonder je aplikacija, s katero otroci sestavljajo program preko uporabe različnih ikon, ki predstavljajo določeno aktivnost robota. Zaporedno povezane ikone tako tvorijo program robota. Program Blockly omogoča programiranje preko sestavljanja blokov v določenem zaporedju. Bloki nam omogočajo, da z njimi ustvarjamo različne algoritme, ki temeljijo na uporabi ukaznih stavkov, pogojušnikov in zank. Reševanje problemov je mogoče z uporabo senzorjev in z izvedbo dogodkov. Aplikacija Xylo je namenjena učenju programiranja preko glasbe in igre. S sestavljanjem glasbenih stavkov se otroci tako učijo ustvarjati algoritme, uporabljati izvršne stavke in zanke. S programom Path programiramo gibanje robota glede na narisano pot, poleg tega lahko robot med opravljanjem poti izvaja dodatne aktivnosti. Aplikacija Go služi za nastavljanje osebnostnih lastnosti robota, kot je njegovo ime, način gibanja, svetlobni in glasbeni učinki [12], [13] ...



Slika 9: Izgled robotov Dash and Dot tablice z aplikacijo Blockly [13].

## 2.2. Robotika v izobraževalnem procesu

Vedno hitrejši razvoj in vpeljava novih tehnologij je posledica napredka in razvoja novih znanj. Pri tem se univerze smatrajo za temeljne nosilke znanj in predstavljajo vez prenosa znanj v podjetja oz. v okolje. Pogosto prenos znanja v okolje ni odvisen le od obstoja univerze v neki regiji, temveč je okolje tisto, ki mora biti dovolj razvito in biti pripravljeno sprejemati novosti. Izkaže se, da sam izobraževalni proces, ki je primarna dejavnost univerz, ni dovolj, da bi se znanja v zadovoljivi meri prenesla v okolje, zato mnoge univerze po svetu organizirajo razne predstavitev in delavnice za otroke in odrasle, s katerimi prenašajo novo znanje v okolje. Prav tako so v času poletnih počitnic organizirane razne poletne šole na temo naravoslovja in tehnike.

Tudi v Sloveniji se vsako leto organizirajo poletne šole in tekmovanja na temo robotov. V različnih kategorijah in težavnostnih nivojih se tekmovanj udeležujejo učenci, dijaki, študenti in učitelji. Najbolj dejavna fakulteta na področju robotskih tekmovanj je Fakulteta za elektrotehniko, računalništvo in informatiko (FERI) Univerze v Mariboru.

Leta 2000 je bilo prvič razpisano tekmovanje, imenovano RoboT2000, in sicer za študente v konstruiranju mini robotov za vožnjo po labirintu. Tekmovanja RoboT so se naslednje leto (2001) lahko že udeležili tudi dijaki. Na podlagi tekmovanj RoboT je bila leta 2004 prvič organizirana robotska liga z imenom RoboLiga, v kateri sodelujejo dijaki in študentje.

Obstajata dve srednješolski robotski tekmovanji, in sicer RoboERŠ na ŠC Velenje in RoboMiš na TŠC Nova Gorica. Osnovno- in srednješolci se lahko udeležujejo tekmovanja ROBObum, ki ga tvorita tekmovanji LEGObum in ROBOsled. LEGObum se deli na dva dela tekmovanj, na LEGObum-8 in LEGObum-9, in temelji na uporabi sestavljanke LEGO Mindstorms. LEGObum-8 je tekmovanje, na katerem morajo roboti v čim krajšem možnem času priti do cilja proge, ki jo predstavlja črna črta na beli podlagi. LEGObum-9 je namenjen izkušnejšim in zajema tekmovanje robotov, pri katerem le-ti izrivajo različne predmete iz črno označenega polja in se izmikajo oviram. Tekmovanje ROBOsled temelji na uporabi sestavljanke, ki je bila v ta namen razvita na fakulteti FERI Univerze v Mariboru.

Tekmovalci sestavljajo komponente kompleta v funkcionalni robot, pri čemer so dovoljene predelave in nadgradnje. Tekmovanje poteka na treh nivojih, in sicer sledenje črti, poznavanje komponent robota in njegovega delovanja ter tekmovanje izvirnosti nadgradenj robota [14].

Med mednarodna tekmovanja spadata tekmovanji FLL (First LEGO League) in RoboCupJunior, na katerem sodelujejo tudi slovenski osnovno- in srednješolci. Sporočilo kratice First (ang. *For Inspiration and Recognition of Science and Technology*) tekmovanja First LEGO League je približevanje in navduševanje mladih nad znanostjo in tehnologijo. Samo ime tekmovanja vsebuje tudi besedo LEGO, ki nam je vsem znana in pod katero si predstavljamo plastične kocke, katerih uporabo omejuje le naša domišljija. Lego kocke predstavljajo osnovne elemente, ki v kombinaciji s tehnologijo Lego Mindstorms na tekmovanju izvršujejo različne naloge na temo iz resničnega življenja. Tako so npr. lansko leto tekmovalci morali razmišljati na temo poti smeti, preprečevanja njihovega nastajanja in ponovne uporabe. Probleme rešujejo od štiri- do desetčlanske ekipe na standardiziranih, nekaj kvadratov velikih, tekmovalnih površinah, vendar tekmovanje ne temelji samo na tekmovalnosti, temveč na sodelovanju in spoštovanju, s čimer mladi pridobivajo tudi socialne veštine [15]. Tekmovanje RoboCupJunior je del svetovnega tekmovanja RoboCup za študente in raziskovalce, vendar je v tej obliki namenjeno osnovno- in srednješolcem. Tekmovanje poteka v treh razredih, in sicer nogomet, reševanje in ples. Kategorija reševanje je postala sestavni del robotskih tekmovanj RoboT in ROBObum [14].

Na področju dela z mladimi je aktivna tudi Fakulteta za strojništvo Univerze v Ljubljani preko poletne šole, imenovane Poletna šola strojništva. V okviru poletne šole organizira laboratorij LAKOS delavnico Mobilni robot, na kateri dijaki razvijajo nizkocenovni mobilni robot z imenom Robošolar. Robošolar je avtonomen robot, ki je zmožen slediti črti ali steni in ga je mogoče voditi na daljavo preko Bluetooth povezave. Pri načrtovanju robota udeleženci uporabljajo za 3D modeliranje program SolidWorks, za načrtovanje elektronskih vezij pa program Altium [16].

Prav tako Fakulteta za elektrotehniko Univerze v Ljubljani ponuja osnovnošolcem in srednješolcem udeležbo na Poletnem taboru inovativnih tehnologij. V sklopu tabora se lahko udeleženci udeležijo različnih delavnic, na katerih se spoznavajo z industrijskimi roboti, mobilnimi roboti, senzorji, načinom delovanja, programiranja itd. [17].



# **3. Vizualno programiranje**

## **3.1. Razvoj programiranja**

Programiranje je postopek, s katerim s pomočjo računalnika rešujemo probleme z uporabo zaporedja predpisanih navodil, pri čemer ima vsako navodilo svojo specifično nalogu. Navodila v celoti predstavljajo algoritem, s katerim pridemo do rešitve problema. Pri programiranju pišemo program, ki ga predstavlja zaporedje stavkov v določenem programskem jeziku. Različni programski jeziki zahtevajo različne nivoje znanja, spremnosti in detajle, ki jih naj bi programer obvladoval.

Edini programski jezik, ki ga lahko računalnik direktno bere je strojni jezik. Prvi računalniški program je bil zato napisan v strojnem jeziku, kar je bila za razvijalce zelo zapletena in utrujajoča naloga. S časom se je programski jezik razvil v zbirni jezik, za katerega je značilna uporaba kombinacij kratkih črkovnih zaporedij. Črkovna zaporedja predstavljajo različice dvojiških zaporedij, njihova sestava pa je razumljiva centralnemu procesorju. Glede na opisan način delovanja strojnega in zbirnega jezika, ju uvrščamo med nizkonivojske programske jezike.

Da bi poenostavili postopek programiranja, so se razvili visokonivojski programski jeziki, ki so uporabniško usmerjeni in prilagojeni programerju. Za višjenivojske jezike je značilna uporaba stavkov v naravnem jeziku, dodani so matematični ukazi in mogoča je izvedba zaporedja več stavkov strojnega jezika z enim ukazom. Ker se v ozadju višjenivojskega jezika ne tvori strojna koda, je potrebno vsak napisan program pretvoriti v strojni jezik. Za izvedbo te naloge imamo na voljo dve možnosti, in sicer uporabo prevajalnika ali tolmača. V zadnjih desetletjih se uveljavlja programski jezik, ki temelji na uporabi grafičnih elementov, katerih kombinacija tvori programsko kodo. Za tovrstno programiranje potrebujemo posebna programska okolja, ki temeljijo na principu »povleci in spusti« (drag and drop) [18].

## **3.2. Predstavitev vizualnega programiranja**

Na vprašanje, kaj je vizualno programiranje, ni mogoče dobiti direktnega odgovora, saj definicija ni točno podana. Tako obstaja več definicij, ki se med seboj razlikujejo. Definicija po Myersu [19] pravi, da se vizualno programiranje nanaša na sistem, ki omogoča

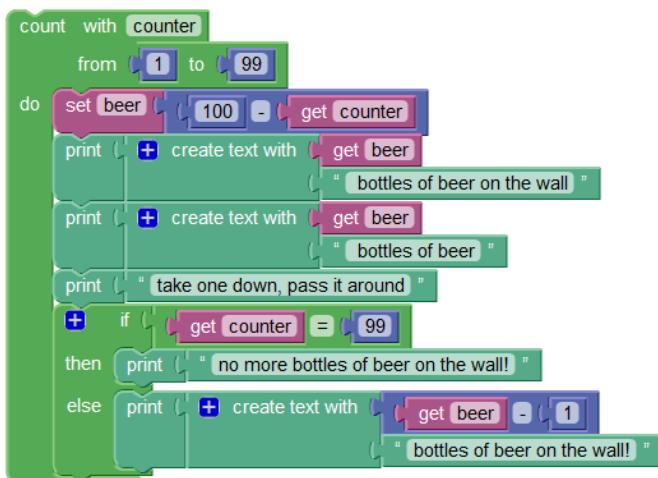
uporabniku programiranje na dvodimenzionalni način. Za razliko od klasičnega tekstovnega programiranja, pri katerem se proces tolmačenja ali prevajanja programa izvede v dolgem enodimenzionalnem toku podatkov. Definicija po Burnettu [19] prav tako omenja uporabo dimenzijs. Vizualno programiranje je po njegovi definiciji programiranje, pri katerem se uporablja za izražanje semantike več kot ena dimenzija. Po Wikipediji [19] pa se definicija vizualnega programskega jezika glasi: Vizualni programski jezik je kateri koli programski jezik, ki omogoča programiranje z manipulacijo grafičnih elementov, namesto uporabe tekstovne kode. Kadar govorimo o vizualnih programskih jezikih, se zelo pogosto uporablja kratica VPL (ang. *Visual Programming Language*).

Klasično programiranje se izvaja preko pisanja tekstovne programske kode, za katero je potrebno integrirano razvojno okolje oz. IDE (ang. *Integrated Development Environment*). Podobno se za vizualno programiranje uporablja programsko okolje imenovano VPE (ang. *Visual Programming Environment*), v katerem s pomočjo grafičnih elementov tvorimo program. Uporaba grafičnih elementov ne zahteva posebnega razumevanja, kar omogoča tudi otrokom, da se prvič pričnejo spoznavati s programiranjem. V nekaterih primerih znanje branja sploh ni potrebo saj programski jezik temelji na uporabi slikovnih simbolov.

```

1 #include<iostream>
2 using namespace std;
3
4 int main() {
5     int number, reverse = 0;
6     cout<<"Input a Number to Reverse: ";
7     cin>> number;
8
9     for( ; number!= 0 ; )
10    {
11        reverse = reverse * 10;
12        reverse = reverse + number%10;
13        number = number/10;
14    }
15    cout<<"New Reversed Number is: "<<reverse;
16
17    return 0;
18 }
```

Slika 10: Primer tekstovne kode v programskem jeziku C++ [20].



Slika 11: Primer blokovnega vizualnega programiranja [21].

Klasično programiranje zahteva od ustvarjalca kode temeljito poznavanje izbranega programskega jezika. Če pogledamo primer kode programskega jezika C++ na sliki 10, vidimo, da je sestavljena iz dveh delov. Prvi del predstavlja glavo programa, drugi pa jedro. V glavi programa se nahajajo povezave na knjižnice kode oziroma zunanje vire npr. #include <iostream>, ki se v našem konkretnem primeru navezuje na ukaza cin in cout v jedru programa. Jedro programa mora v programskem jeziku vedno vsebovati funkcijo main (glavna funkcija), ki kliče druge funkcije. Klicana funkcija izvaja po zaporedju svoje stavke. Stavki kode se nahajajo med zavitima oklepajema {}, kar tvori telo klicane funkcije. V primeru kode na sliki 10 funkcij ni vidnih, vendar imajo enako zgradbo kot glavna funkcija main. V funkciji main je vidna for zanka, ki spada v razred zank, pri čemer vsaka od zank izvaja svojo funkcijo. Tako napisana programska koda še z ostalimi elementi programskega jezika tvori končni program, ki se izvaja sekvenčno [22].

Za vizualno programiranje potrebujemo primerno programsko okolje, v katerem s pomočjo že definiranih grafičnih oblik gradimo programsko kodo. Pri vizualnem programiranju, katerega primer je prikazan na sliki 11, je princip sekvenčnega izvajanja programa enak klasičnemu, le da tekstovne funkcije oziroma ukaze zamenjujejo grafični oblački, bloki ali katera druga grafična oblika. Vizualno programiranje poteka s sestavljanjem grafičnih oblik, katerim nastavljamo parametre, v ozadju pa se tako ustvarja programska koda. Uporabnik lahko generira za svoje potrebe tudi svoje grafične oblike.

### 3.3. Področja uporabe

Glavni namen večine vizualnih programskih jezikov je približati programiranje širši množici, oziroma uporabnikom olajšati delo s programi. Obstaja veliko različnih področij uporabe vizualnih programskih jezikov, zato bodo v nadaljevanju našteta le nekatera področja uporabe in primeri programov:

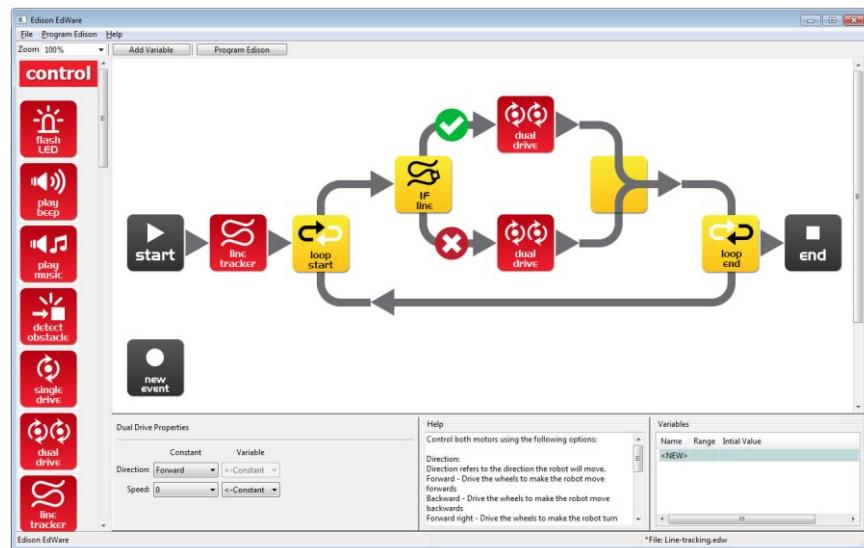
- izobraževanja programiranja in robotike (Scratch, Blockly, Ardublock, Minibloq, Modkit, Ardublockly, NETLab tool ...),
- 3D in 2D animacije ter ustvarjanje video iger (Mama, Hopscotch, Construct 2),
- razvoj aplikacij na pametnih telefonih, tablicah in računalnikih (Microsoft touch develop),
- simulacije in modeliranje (Simulink),
- testiranje, meritve in nadzor (LabVIEW),
- urejanje avdio in video posnetkov (DaVinci resolve 14),
- obdelava podatkov (RTMaps),
- modeliranje umetne inteligence (Spirops AI, Angryant behave),
- razvoj računalniške grafike (Quartz Composer),
- filmska industrija (Golaem crowds behavior),
- ... [23].

### 3.4. Delitev vizualnih programskih jezikov

Za različne namene in področja uporabe so bila razvita različna vizualna programska okolja. Jezike za vizualno programiranje lahko delimo glede na tip uporabljenih vizualnih elementov, in sicer na:

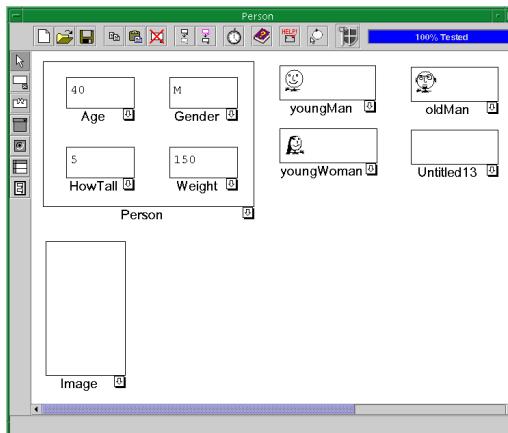
- jezike, ki temeljijo na ikonah,
- jezike, ki temeljijo na obrazcih in
- jezike, ki temeljijo na diagramih.

Programski jezik, ki temelji na ikonah, je vizualni programski jezik, pri katerem uporabljamo ikone kot osnovne gradnike programa. Programiranje poteka s postavljanjem ikon na določeno lokacijo na delovni površini programa, pri čemer dodatni pomen izhaja iz relativnega položaja, povezav in lastnosti posameznih objektov oz. ikon. Skupina povezanih ikon tako predstavlja program, kar prikazuje slika 12. Tovrstni jeziki so primerni za naloge preučevanja in obdelave baz podatkov.



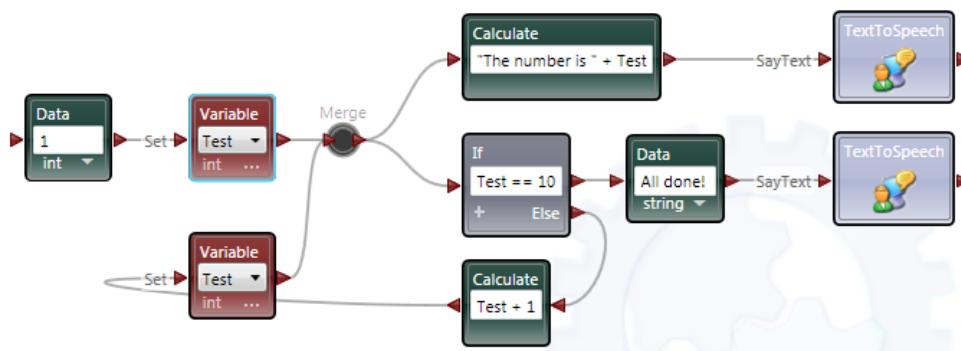
Slika 12: Prikaz programiranja z uporabo ikon [24].

Programi, ki temeljijo na obrazcih, imajo podobno upodobitev in način programiranja kot preglednice. Programiranje predstavlja časovno spremenjanje skupine medsebojno povezanih celic in pogosto omogoča programerju vizualizacijo izvršitve programa, ki je vidna kot časovno spremenjanje zaporedja različnih stanj celic. Na sliki je prikaz programa Forms/3, ki ponazarja programiranje s pomočjo uporabe obrazcev.



Slika 13: Prikaz programiranja z uporabo obrazcev (Forms/3) [25].

Danes najbolj uporabljen tip vizualnega programskega jezika temelji na ustvarjanju diagramov, kar predstavlja končni program. Najpogosteje uporabljeni obliki so t. i. diagrami toka. Posamezni objekti so na delovni površini vizualnega okolja medsebojno smerno povezani, kar daje programu smer toka podatkov. Tovrstni programski jezik vsebuje značilnosti ostalih vizualnih programskih jezikov, saj določujemo lastnosti posameznih objektov z uporabo obrazcev. Zaradi specifike omenjenih programskih jezikov in njihovega načina zaporednega izvajanja programa je velikokrat mogoče spremeniti besedilne programske jezike v obliko vizualnega programskega jezika. Uporaba diagramskega vizualnega jezika je posebej značilna za modeliranje programov, ki temeljijo na toku podatkov [26].



Slika 14: Prikaz programiranja z uporabo diagramov [27].

Obstaja tudi delitev glede na tipe vizualnih elementov; tako vizualne programske jezike delimo na:

- čisto vizualne,
- hibridne (kombinacija besedila in grafičnih elementov) in
- ostale sisteme (sistemi z uporabo omejitve in obrazcev).

Osnovni gradniki čisto vizualnih programskih jezikov so ikone, ki jih uporabljamo za programiranje. Po koncu programiranja poteka preverjanje in izvrševanje programa v istem vizualnem okolju. Program je preveden neposredno iz vizualne upodobitve, pri čemer se prevajanje programa nikoli ne izvaja preko vmesne besedilne faze.

Hibridne sisteme delimo na dve oblike, in sicer na hibridne sisteme, v katerih programiramo vizualno, nato pa so prevedeni v osnovni višenivojski tekstovni jezik. Druga oblika predstavlja sisteme, ki vključujejo grafične elemente v tekstovnem jeziku.

Sisteme z uporabo omejitev pa uporabljamo, ko modeliramo fizične objekte kot objekte, ki so predmet omejevanja z namenom posnemanja naravnih zakonov (npr. gravitacija) [28], [29], [30].

### 3.5. Prednosti in pomanjkljivosti

Največja prednost vizualnih programskega jezika pred tekstualnim je manjše število sintaktičnih elementov, ki jih mora uporabnik usvojiti za začetek programiranja. Uporabnik se tako ni prisiljen učiti programskih izrazov za podajanje programskih elementov, temveč so mu elementi neposredno predstavljeni. Elementi so tako predstavljeni v grafični obliki, kar uporabniku omogoča vizualizacijo programske logike, to pa je primerno zlasti za začetnike, ki se na ta način srečujejo s spoznavanjem povezav med logičnimi programskimi strukturami. Grafični elementi so že v naprej definirani, zato se verjetnost za pojav napake pri ustvarjanju programske kode zmanjša. Način programiranja z grafičnimi elementi omogoča programiranje v situacijah, v katerih nimamo možnosti uporabe tipkovnice.

Pri postavljanju grafičnih blokov na delovno površino programa le-ti zavzamejo veliko prostora, zato postaja program, kot celota, nepregleden. V ta namen se programiranja obsežnih programov z uporabo vizualnih programskega jezika izogibamo. Majhne ali velike spremembe v primeru velikih programov so včasih težko izvedljive, ne da bi pri tem močno vplivali na vizualni izgled programa, kar na koncu od uporabnika zahteva nepotrebno dodatno delo [26], [31].

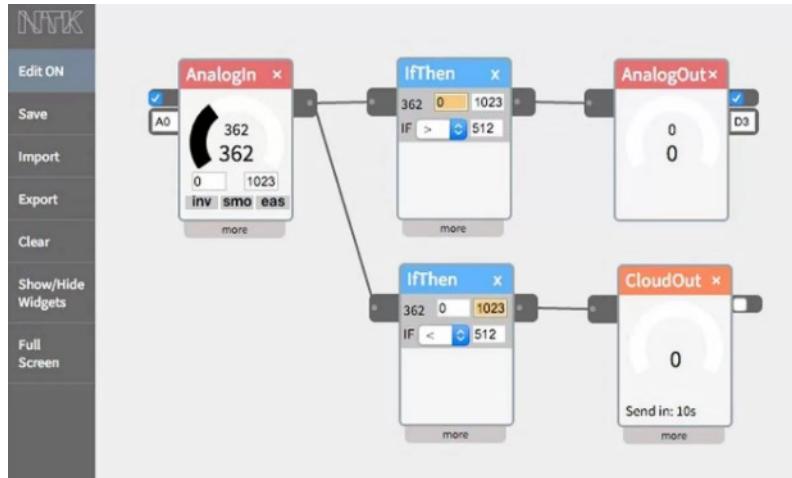
## **4. Pregled stanja in izbor vizualnega okolja**

Na svetovnem spletu se nahaja veliko različnih vizualnih programskih okolij, ki so prilagojena izvajanju specifičnih nalog za različne namene uporabe. Za naše potrebe bomo posvetili pozornost programskim okoljem z možnostjo programiranja senzorjev in vgrajenih računalnikov. Tovrstna programska okolja so po navadi odprtakodna in brezplačna.

### **4.1. NETLab Toolkit**

NETLab Toolkit oz. krajše NTK je sistem za ustvarjanje sočasnega vizualnega vmesnika za upravljanje in programiranje mikrokrmilniških platform. NTK je sestavljen iz dveh delov, in sicer iz grafičnih blokov oz. pripomočkov (Widges) in podatkovnega središča (Hub). Pripomočki predstavljajo vizualni vmesnik in vsebujejo vhodna polja, drsниke, gumbe ter ostale nadzorne elemente, zato navadno sprejemajo vrednosti, jih preko določenega procesa transformirajo in pošljejo do naslednjega pripomočka. Podatkovno središče je strežniška aplikacija in skrbi za usklajevanje komunikacije med pripomočki in zunanjimi napravami. Postavljanje pripomočkov na našem delovnem prostoru temelji na principu povleci in spusti (drag and drop). Posamezni pripomočki imajo različne funkcije, zato preko uporabe kombinacij različnih pripomočkov in njihovih medsebojnih povezav ustvarimo nov program. NTK programi so kompatibilni z Arduino in Linux strojno opremo (Raspi, Galileo).

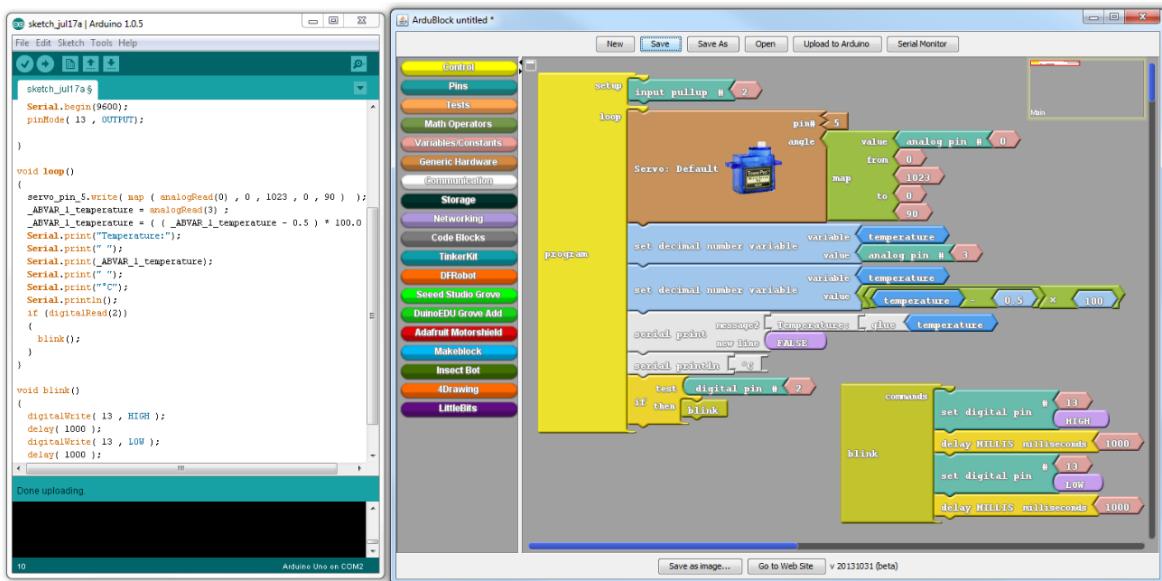
Za namestitev NTK programskega okolja je potrebno na računalnik namestiti pripomočke (Widges) in podatkovno središče (Hub). Poleg tega je potrebna namestitev Arduino razvojnega okolja, v katerem se povežemo z računalnikom na Arduino platformo z uporabo Firmata knjižnice, ki jo moramo prav tako prenesti s spletja in jo namestiti na Arduino programsko okolje. [32].



Slika 15: Izgled vizualnega programskega okolja NTK [32].

## 4.2. Ardublock

Ardublock je odprtokodni programski jezik za programiranje Arduina s pomočjo grafičnih elementov oz. z uporabo blokov, pri čemer programiranje poteka preko uporabe metode povleci in spusti (drag and drop block). Ardublock je programska oprema in predstavlja vtičnik razvojnemu okolju Arduino ter omogoča uporabniku programiranje z uporabo funkcij, zapisanih v blokih. Pozitivna stran Ardublocka je generacija vrstic kode, ki so dobesedni prevod sestavljenih blokov. V Arduino knjižnici se nahajajo že definirani bloki, s katerimi je mogoče upravljati raznolike elektronske komponente, ki so podprte s strani proizvajalcev kompletov elektronske opreme, kot so Scoop, Adafruit, DFrobot, TinkerKit. Za namestitev Ardublocka je potrebna namestitev Arduino razvojnega okolja in nastavitev povezave med lokacijo namestitve Ardublocka v programu Arduino [33].

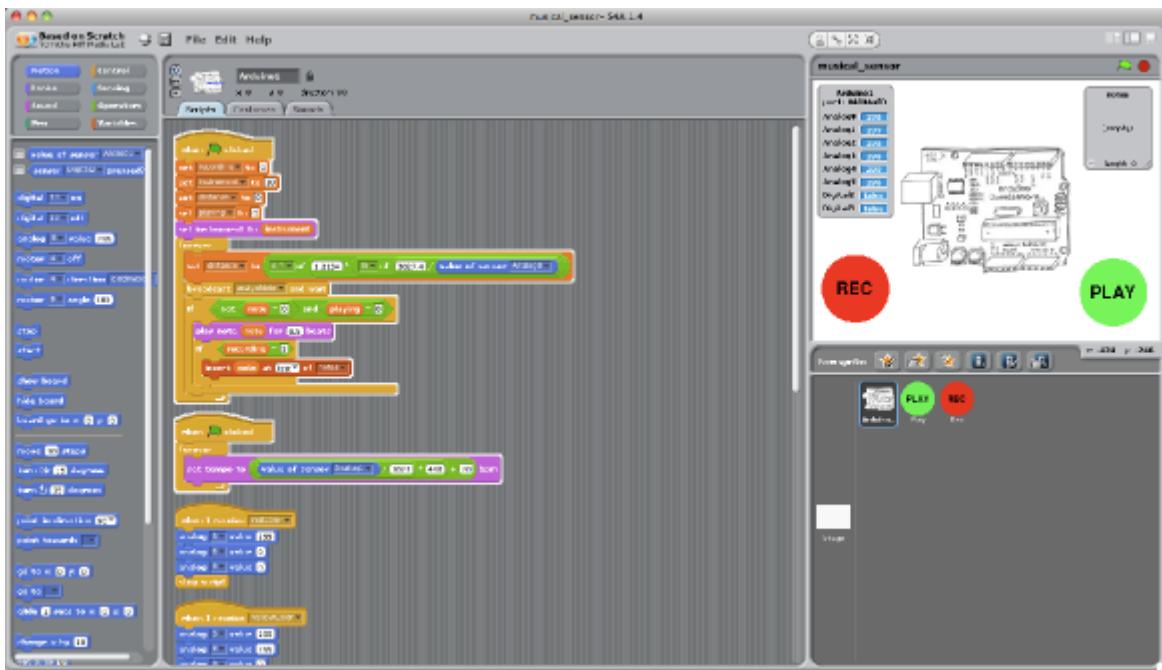


Slika 16: Izgled vizualnega programskega okolja Ardublock [33].

## 4.3. S4A Scratch

Scratch je odprtokodni programski jezik in spletna skupnost, kjer lahko otroci programirajo in delijo interaktivne medije, kot so zgodbe, igre in animacije z ljudmi z vsega sveta. Ko otroci ustvarjajo s Scratchom, se učijo ustvarjalnega mišljenja, skupinskega dela in sistematičnega argumentiranja. Scratch je ustvarjen in ga ohranja ter vzdržuje skupina Lifelong Kindergarten group iz MIT Media Laboratorija.

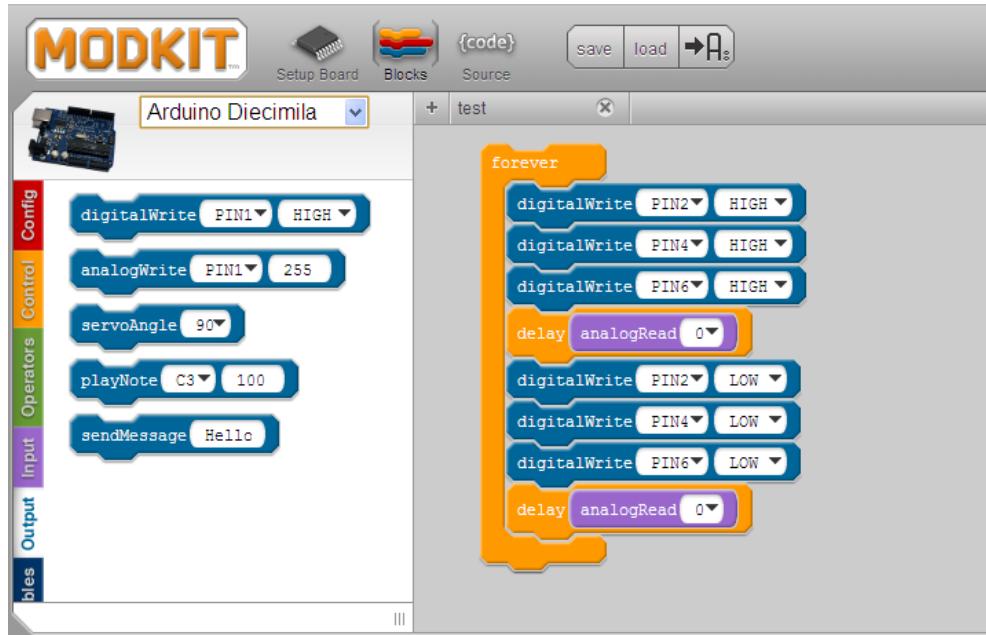
S4A Scratch je nadgradnja programa Scratch z možnostjo enostavnega programiranja Arduino odprtokodne platforme. Program zagotavlja nove bloke za upravljanje senzorjev in aktuatorjev, priključenih na Arduino. Glavni cilj projekta je privabiti ljudi k programiranju in zagotoviti enostavnejše delo glede na medsebojno uporabo različnih kombinacij Arduino platform. Program ponuja bloke osnovnih mikrokrmlniških funkcij in blokov vnosa in izpisa. S4A deluje na Arduino platformah Diecimila, Duemilanove in Uno. Glede na vire delovanje ostalih platform ni bilo preizkušeno. Delovanje na ostalih platformah je mogoče, vendar še ni bilo preizkušeno [34].



Slika 17: Izgled vizualnega okolja S4A Scratch [34].

## 4.4. Modkit

Z uporabo programa Modkit lahko programiramo Arduino in Arduino kompatibilno strojno opremo z uporabo grafičnih blokov, katerih uporaba je prevzeta iz Scratch razvojnega okolja. Podobno kot Scratch ima program že ustvarjene bloke, prilagojene upravljanju mikrokrmlniških Arduino platform. Modkit ni odprtokodna programska oprema, zato ustvarjanje novih blokov ni mogoče [35].

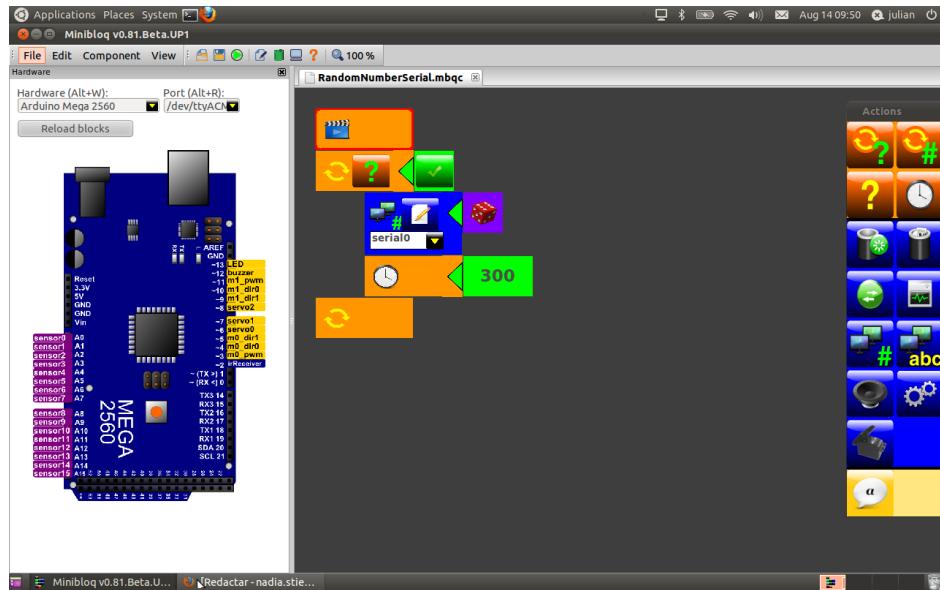


Slika 18: Izgled vizualnega okolja Modkit [35].

## 4.5. MiniBloq

MiniBloq je vizualno programsko okolje za Arduino, Multiplo in ostale mikrokrmlniške platforme ter robote. Glavni cilj programa je približati mikrokrmlniške in robotske platforme otrokom in začetnikom programiranja. Ne glede na operacijski sistem so za delovanje platform potrebni gonilniki, ki jih MiniBloq preko njegove namestitve na računalnik že vsebuje. Če želimo zagnati program MiniBloq na operacijskem sistemu Windows, moramo prenesti Windows installer version, ki nam namesti program na naš računalnik. Za komunikacijo med računalnikom in platformo, ki jo želimo programirati, moramo namestiti pripadajoče gonilnike.

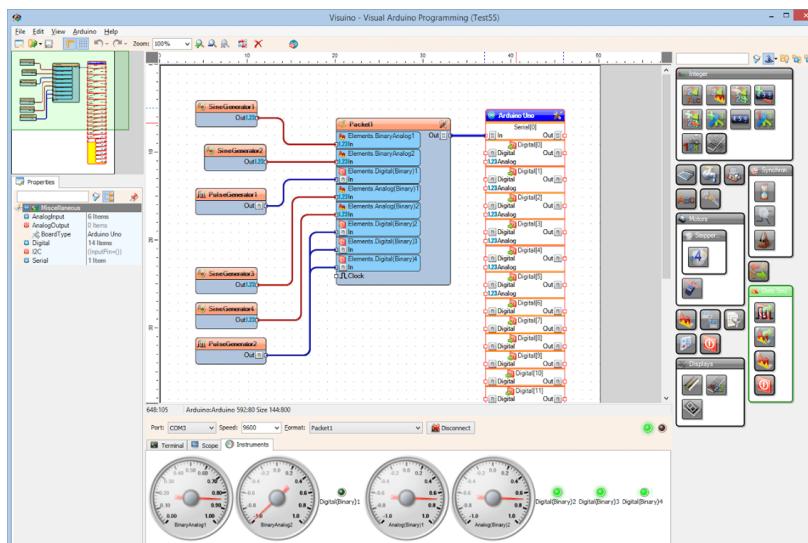
Prednost uporabe MiniBloqa je njegova enostavnost, kar pomeni, da lahko naš prvi program ustvarimo le z nekaj kliki. Sočasni generator kode nam ustvarja kodo, med tem ko dodajamo bloke na naše delovno okolje oz. spreminjam vrednosti parametrov. Na ta način MiniBloq olajšuje uporabnikom prehod iz vizualnega na tekstovno programiranje. Vgrajena funkcija pregledovanja ustreznosti kode uporabnikom olajšuje iskanje napak in s tem programiranje. Program omogoča manipulacijo blokov s principom povleci in spusti, kopiranje in lepljenje, povečavo delovnega prostora ter ostale uporabne pomočke. Vgrajeni terminal uporabnikom omogoča pošiljanje in sprejemanje podatkov preko USB priključka. Paket programa MiniBloq vsebuje vse, kar je potrebno za začetek programiranja, poleg tega je prenosljiv, kar pomeni, da ne potrebuje namestitve na računalnik. Potrebna je namestitev gonilnikov, če želimo programirati platformo, kot je npr. Arduino. Hitro delovanje programa omogoča uporabo tudi na manj zmogljivih računalnikih. MiniBloq je modularen in razširljiv, kar omogoča ustvarjanje novih blokov [36].



Slika 19: Izgled vizualnega okolja MiniBloq [36].

## 4.6. Visuino

Visuino je najnovejše vizualno programsko okolje podjetja Mitov Software. Programsko okolje nam omogoča programiranje Arduinovih in ostalih kopij Arduino plošč. Elemente, ki jih najdemo v Visuino programski opremi, predstavljajo dejanske komponente strojne opreme, katerih grafične upodobitve lahko enostavno zlagamo v program z načinom povleci in spusti. Za delovanje programa ni potrebna nobena namestitev strojne ali katere koli druge opreme. Po uspešno ustvarjenem programu lahko priključimo Arduino ploščico na računalnik in na njo naložimo program. Program se ustvarja z zlaganjem in povezovanjem grafičnih elementov v diagrame zaradi česar je uporaba programa enostavna in pregledna za uporabnika. Grafični elementi in način programiranja s postopkom povleci in spusti temeljijo na tehnologiji OpenWire. OpenWire je platforma, ki omogoča razvijalcem ustvariti kompleksen program brez kakršnega koli pisanja programske kode [37].



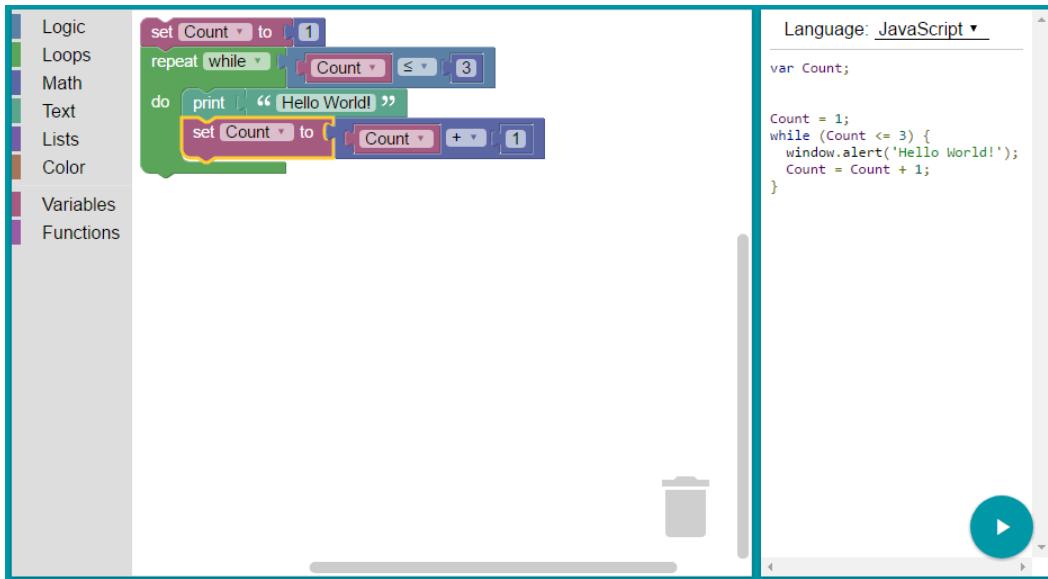
Slika 20: Izgled vizualnega okolja Visuino [37].

## 4.7. Blockly

Blockly je knjižnica, s pomočjo katere lahko dostopamo do vizualnega urejevalnika kode na spletu in Android aplikacijah. Blockly je projekt podjetja Google, pri čemer njegova podoba temelji na izgledu Scratch vizualnega programskega okolja. Zopet imamo na razpolago knjižnico že definiranih blokov. Iz knjižnice že definiranih blokov lahko z njihovim zlaganjem v logičnem zaporedju dobimo programsko kodo v JavaScript, Python, PHP ali Dart programskem jeziku.

Blocklyjev vmesnik sestoji iz nabora orodij, ki so razdeljena na kategorije glede na njihove funkcije. V posameznih kategorijah se nahajajo že definirani bloki. Ker je Blockly odprtokodni sistem, je mogoče ustvariti nove bloke po naših potrebah in si tako personalizirati naše delovno okolje.

Novo ustvarjeni bloki potrebujejo definicijo bloka in njegov generator kode, pri čemer definicija bloka predpisuje izgled bloka, generator kode pa prevod bloka v izvršilno kodo. V pomoč pri ustvarjanju novih blokov imamo na voljo Googlovo orodje Block factory, ki uporabniku preko uporabe že definiranih blokov olajša konstrukcijo novih blokov. Blockly nam ne omogoča direktnega programiranja Arduino platforme, vendar nam služi kot orodje za izdelavo novih blokov funkcij preko vizualnega vmesnika Block factory, kar nepoznavalcem programskih jezikov zelo olajša ustvarjanje novih blokov. Generirano kodo definicije novega bloka lahko tako prenesemo v izvorno kodo odprtokodnih vizualnih programskih okolij, ki podpirajo programiranje Arduino platform.



Slika 21: Izgled vizualnega okolja Blockly [38].

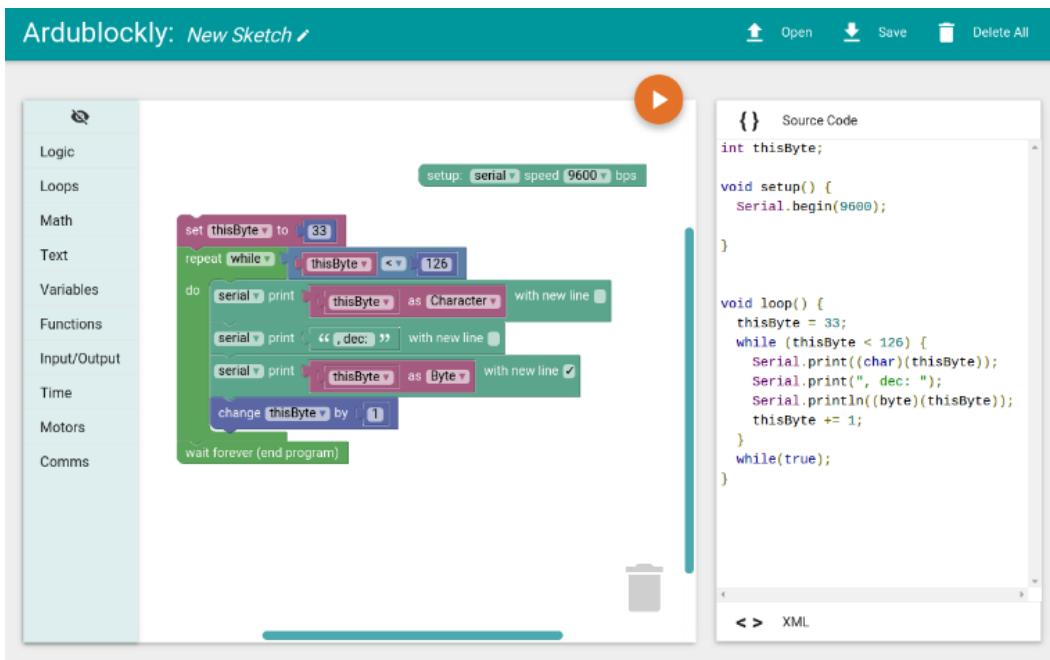
## 4.8. Ardublockly

Ardublockly je vizualno programsko okolje, prilagojeno programiranju Arduino platforme, in temelji na razvojnem okolju Google Blockly, katerega predelava je bila izvedena z namenom ustvarjanja Arduino kode.

Z Ardublocklyjem lahko ustvarjamo Arduino kodo preko uporabe vizualnega principa, s katerim bloke kode sestavljamo na principu povleci in spusti (drag and drop block). Z omenjenim programom lahko naložimo generirano kodo direktno na Arduino platformo, pri čemer nas program tudi opozarja na napake pri programiranju.

Ardublockly je kompatibilen s širokim naborom Arduino platform in deluje na operacijskih sistemih Windows, Linux in Mac OS X. Razvojno okolje Ardublockly je še vedno v razvoju, zato obstaja na spletu seznam stvari, ki jih bodo razvijalci v prihodnosti še dodali.

Za uporabo Ardublocklyja ni potrebna njegova namestitev na osebni računalnik, saj programiranje lahko izvedemo v spletni demo verziji. Spletna verzija nam ne omogoča nalaganja generirane kode na Arduino platformo, za kar potrebujemo popolno namestitev Ardublockly programa na naš osebni računalnik [39].



Slika 22: Izgled vizualnega okolja Ardublockly [39].

## 4.9. Izbor vizualnega okolja

Na podlagi pregleda vizualnih programskih okolji in njihovih opisov smo spoznali pozitivne in negativne strani opisanih programov ter možnosti njihove nadgradnje. Za naše potrebe smo izvedeli selekcijo programov, pri čemer smo se osredotočili na to, da izbran program omogoča programiranje našega robota (Arduino platforme) in da je čim bolj prijazen do uporabnika. Ker je naša naloga ustvariti nove funkcije in prenos programov na mobilni robot, smo pri izbiri ustreznega vizualnega okolja upoštevali omenjena kriterija.

Program NETLab Toolkit je zelo uporabno orodje za sočasno upravljanje Arduino platforme, vendar ima kompleksen vmesnik s prevelikim številom parametrov, ki so za naše potrebe programiranja mobilnega robota nepotrebne. Program Ardublock potrebuje za programiranje Arduino platforme še namestitev programa Arduino, kar ne predstavlja posebne omejitve, zato smo ga uvrstili na seznam kandidatov za izbor. Program S4A smo

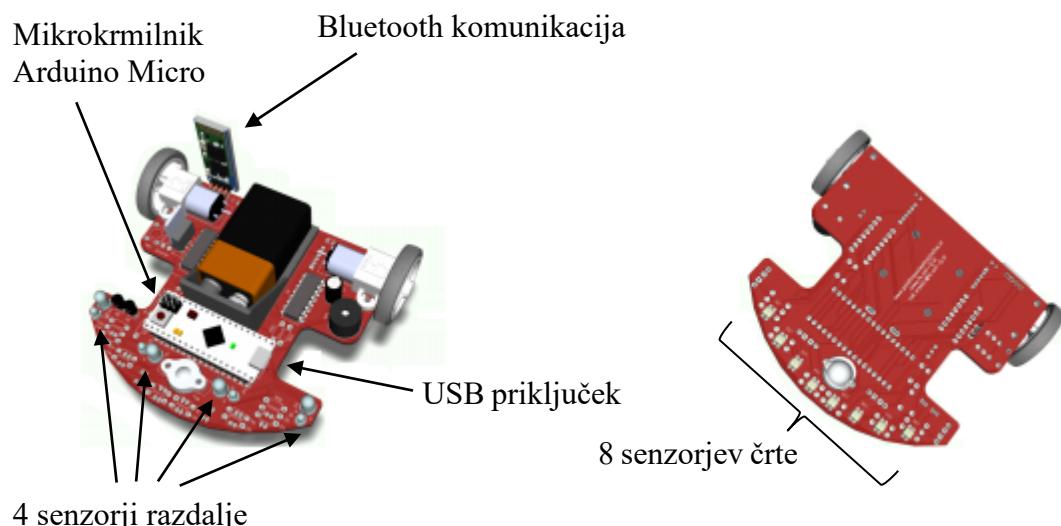
opustili, saj zahteva uporabo dodatne knjižnice Firmata za komunikacijo z Arduinom, zato je iz tega vidika njegova povezljivost zahtevnejša za razliko od programa Ardublock. Program Modkit ima prijazen vmesnik in ima dobro povezljivost z Arduino platformo, pri čemer je njegova slaba lastnost da ni odprtokoden in tako ne omogoča vnosa novih blokov. Program MiniBloq omogoča enostavno namestitev in uporabo, vendar je potrebno za komunikacijo med računalnikom in določeno platformo namestiti ustreerne gonilnike, zato je bil odstranjen iz nabora. Visuino je izrecno namenjen programiranju Arduino platforme in temelji na načinu ustvarjanja programov preko medsebojnega povezovanja blokov. Na ta način dobimo končni program v obliki diagrama, kar za naše potrebe ni sprejemljivo, saj želimo ustvarjati program z zlaganjem grafičnih blokov kode v logičnem zaporedju. Blockly ne omogoča direktnega programiranja Arduino platforme, lahko pa nam služi kot uporabno orodje pri ustvarjanju novih blokov funkcij. Ardublockly ne potrebuje dodatne namestitve programa Arduino in omogoča neposredni vpogled in prenos generirane kode na Arduinovo platformo. Program vsebuje orodjarno z že definiranimi bloki kode, ki se nahajajo v posamezni kategoriji. Z omenjeno razporeditvijo blokov kode po posamezni kategoriji je program bolj pregleden. Preglednost povečuje tudi sočasno generiranje Arduino kode glede na ustvarjen program, ki jo lahko vidimo v posebnem oknu na desni strani programskega okolja. Poleg vseh naštetih ugotovitev ima Ardublockly enostaven in prijazen izgled uporabniškega vmesnika, kar omogoča uporabniku enostavnejšo uporabo.

Po končani izvedbi selekcijanja in preučevanju naštetih programskih okolij smo se odločili za uporabo programskega okolja Ardublockly. Programske okolje omogoča preprost način ustvarjanja novih blokov za krmiljenje našega mobilnega robota, saj si lahko pomagamo z Googlovim orodjem Block factory. Na ta način dobimo kodo oblike in kodo bloka, ki jih nato lahko naknadno tekstovno prilagodimo za naše potrebe. Poleg tega pa je programske okolje enostavno in pregledno, kar olajša uporabniku delo s programom.

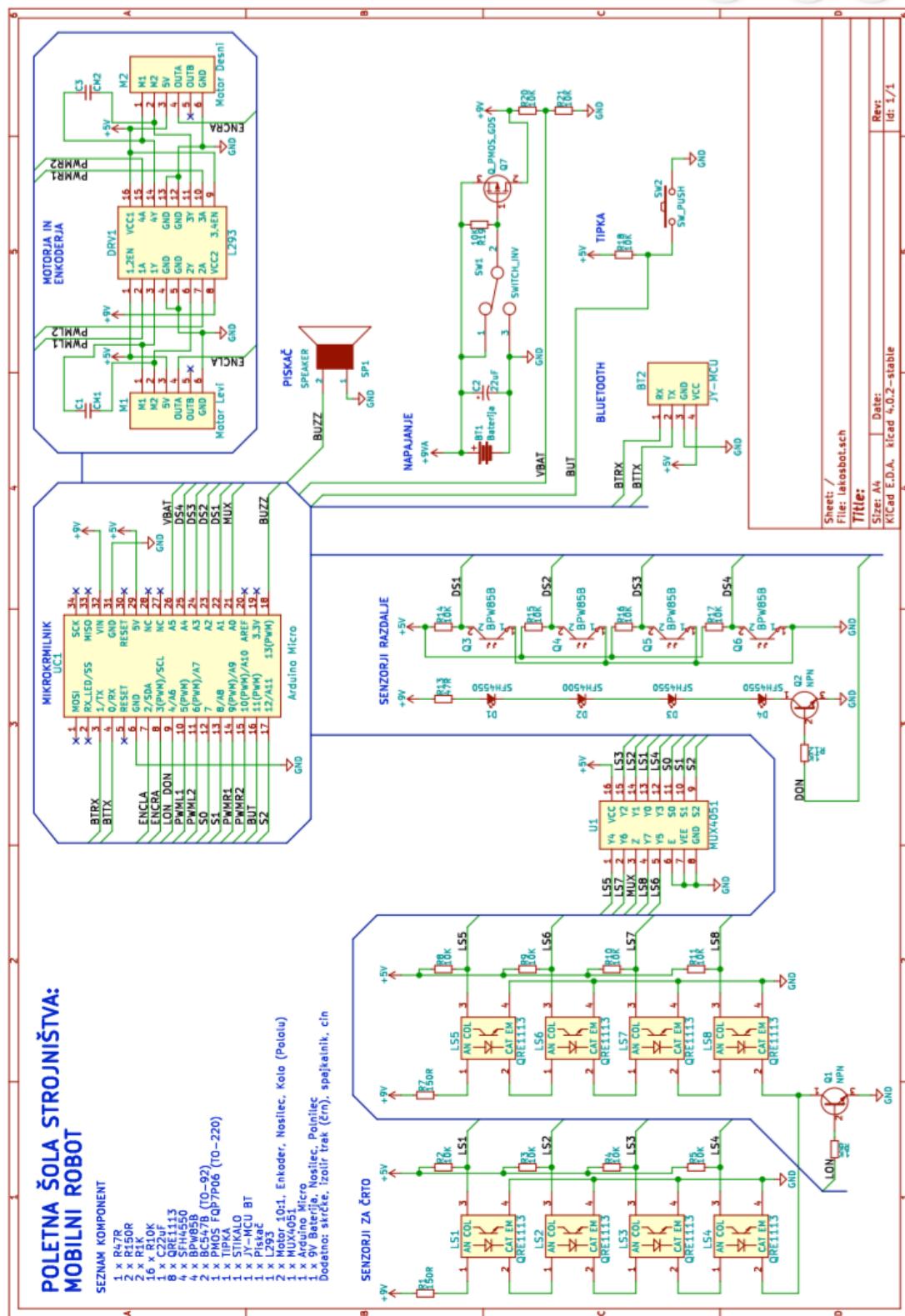
# 5. Opis mobilnega robota

## 5.1. Zgradba mobilnega robota

Srce mobilnega robota predstavlja Arduino Micro plošča, ki skrbi za izvajanje vseh funkcij robota in je pritrjena na osnovno ploščo s tiskanim vezjem. Osnovna plošča služi za napajanje in prenos podatkov med posameznimi gradniki kot tudi za njihovo medsebojno fizično povezavo. Za pogon posameznega kolesa skrbi motor z enkoderjem, ki izvaja meritve zasukov in določa smer vrtenja kolesa. Tako je preko dveh neodvisnih motorjev zagotovljeno poljubno gibanje robota. Neposredne trke z ovirami preprečujejo štirje senzorji razdalje, ostalih osem senzorjev pa omogočajo robotu, da lahko sledi črti. Za komunikacijo s krmilnikom se lahko z računalnikom povežemo preko USB priključka ali Bluetooth komunikacije. Mobilni robot v celoti napaja 9V baterija, ki se nahaja na lokaciji med obema kolesoma in tako tudi zagotavlja dodatno stabilnost robota. Za oddajanje zvočnih signalov skrbi piskač. Na sliki 24 so prikazane medsebojne povezave med gradniki mobilnega robota [16].



Slika 23: Mobilni robot zgornji del (levo) in spodnji del (desno) [41].



Slika 24: Prikaz vezalne sheme mobilnega robota [41].

## 5.2. Funkcije mobilnega robota

Gibanje mobilnega robota je odvisno od naloženega programa na mikrokrmlnik. Program se napiše v integriranem razvojnem okolju z uporabo različnih programskih funkcij in se nato prenese na mikrokrmlnik. Mikrokrmlnik ta program bere in pošilja ukaze na druge gradnike robota, kar omogoča njegovo delovanje. Za programiranje našega robota je potreben program Arduino, preko katerega s pomočjo USB vodnika ali Bluetooth komunikacije prenesemo generirano kodo na mikrokrmlnik (Arduino Micro).

Za delovanje mobilnega robota so potrebne programske funkcije, ki se za primer Arduino ploščice programirajo v programske jeziku C++. Na podlagi spodaj navedenih funkcij, ki smo jih pridobili kot vhodni podatek, smo nadgradili naše vizualno okolje z naslednjimi funkcijami:

- void pelji(int hitrost, int rotacija);
- void peljiLD(int smerLevo, int hitrostLevo, int smerDesno, int hitrostDesno);
- void kalibrirajCrto(int trajanje);
- int crta();
- void cakajTipko();
- int baterija();
- void zvocnik(int frekvenca, int trajanje);
- attachinterrupt(digitalPinToInterruption (pin), ISR, mode);

Funkcijo attachinterrupt smo sprogramirali le v JavaScriptu, saj ni bila podana kot vhodni podatek kot vse ostale zgoraj naštete funkcije. Na sliki 25 je prikazana programska koda za funkcijo attachinterrupt.

```

1 Blockly.Arduino['attachInterrupt'] = function(block) {
2   // Construct code
3
4   var branch = Blockly.Arduino.statementToCode(block, 'attachInterrupt');
5   var number_pin = block.getFieldValue('pin')||1;
6   var text_mode = block.getFieldValue('mode');
7
8   var fname=block.getFieldValue('funkcija');
9   Blockly.Arduino.addSetup(fname+"_","attachInterrupt("+number_pin+","+fname+","+text_mode+")", true);
10  Blockly.Arduino.userFunctions_[fname] = "void "+fname+"() { \n" + branch + "\n }";
11  return null;
12 };
13

```

Slika 25: Izgled JavaScript programske kode za funkcijo attachinterrupt.

Vse naštete funkcije so sprogramirane v datoteki Robot.cpp in napisane v programske jeziku C++. V prilogi se nahaja celotna koda vseh naštetih funkcij mobilnega robota. Za razliko od ostalih naštetih funkcij se funkcija attachinterrupt nahaja v knjižnici Arduino programskega okolja, zato je ni bilo potrebno programirati. Potrebno jo je bilo samo preklicati z njenim imenom v glavni zanki programa setup (vrstica 9, slika 25.) z vsemi njenimi vhodnimi parametri. Detajnejši opis omenjene funkcije (kot tudi ostalih funkcij) je predstavljen v poglavju 5.2 Funkcije mobilnega robota.



## 6. Nadgradnja vizualnega okolja

V poglavju 4 Pregled stanja in izbor vizualnega okolja smo izbrali programsko okolje Ardublockly. Za namestitev programskega okolja Ardublockly, ki ga bomo modificirali, moramo najprej na osebni računalnik namestiti Git. Git je sistem za upravljanje z izvorno kodo, in velja za najbolj razširjen sistem na področju razvoja programske opreme [42].

Ardublockly naložimo s pomočjo programa Git, in sicer v vrstico z želeno lokacijo namestitve izvorne kode programa vpišemo

```
git clone https://github.com/carlosperate/ardublockly.git
```

Omenjen zgornji ukaz nam ustvari kopijo programskih datotek. V mapi ardublockly moramo inicializirati datoteko closure library in ardublockly.wiki z ukazom

```
git submodule update --init --recursive
```

Na ta način smo si pripravili program za nadaljnjo nadgradnjo .

### 6.1. Modifikacija liste orodij (Toolbox)

Začetek modifikacije liste orodij se prične z dekomprimiranjem HTML kode v datoteki index.html na lokaciji \ardublockly\ardublockly.

```
<!-- Ardublockly - These three files contain the compress version -->
<script src="../blockly/blockly_compressed.js"></script>
<script src="../blockly/blocks_compressed.js"></script>
<script src="../blockly/arduino_compressed.js"></script>
<!-- To use the uncompressed version comment out the above and comment in the ones below
<!--<script src="../blockly/blockly_uncompressed.js"></script>
<script src="../blockly/blocks/logic.js"></script>
<script src="../blockly/blocks-loops.js"></script>
<script src="../blockly/blocks/math.js"></script>
<script src="../blockly/blocks/text.js"></script>
<script src="../blockly/blocks/lists.js"></script>
```

Slika 26: Izgled komprimirane kode.

Kodo dekomprimiramo z odstranitvijo znakov za komentiranje <!-- -->, ki se nahajata pred in za sivo obarvanim besedilom spodnjih stavkov na sliki 26. Zakomentirati pa moramo tri zgornje stavke besedila, ki vsebujejo besedo compressed. Končni rezultat dekomprimirne kode je prikazan na sliki 27.

```
<!-- Ardublockly - These three files contain the compress version -->
<!--script src="../blockly/blockly_compressed.js"></script>
<script src="../blockly/blocks_compressed.js"></script>
<script src="../blockly/arduino_compressed.js"></script-->
<!-- To use the uncompressed version comment out the above and comment in the ones below
<script src="../blockly/blockly_uncompressed.js"></script>
<script src="../blockly/blocks/logic.js"></script>
<script src="../blockly/blocks/loops.js"></script>
<script src="../blockly/blocks/math.js"></script>
<script src="../blockly/blocks/text.js"></script>
<script src="../blockly/blocks/lists.js"></script>
```

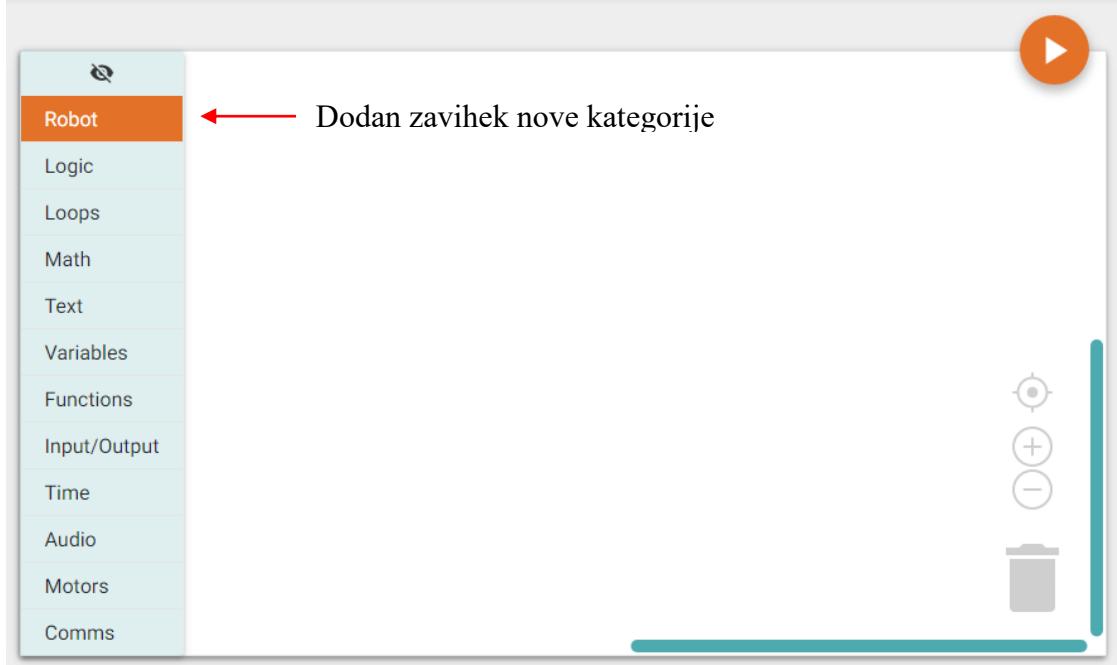
Slika 27: Izgled dekomprimirne kode.

Listo orodij smo modificirali z razširitvijo obstoječe liste orodij s kategorijo, imenovano Robot. V obliki blokov se bodo v novi kategoriji Robot nahajale vse nove funkcije, ki so potrebne za delovanje robota. Na lokaciji \ardublockly\ardublockly spremenimo datoteko z imenom ardublockly\_toolbox.js z dodatkom kode, ki jo prikazuje slika 28.

```
Ardublockly.TOOLBOX_XML =
'<xml>' +
'  <sep></sep>' +
'  <category id="robot" name="Robot">' +
'    </category>' +
'  <sep></sep>' +
```

Slika 28: Dodatek xml kode za razširitev obstoječe liste orodij.

Dodana koda v datoteki ardublockly\_toolbox.js vpliva na spremembo izgleda vizualnega okolja v brskalniku.



Slika 29: Dodana kategorija kot rezultat spremembe xml kode.

## 6.2. Ustvarjanje novega bloka

Za ustvarjanje novega bloka moramo v prvem koraku definirati blok in nato v drugem koraku njegovo generirajočo kodo. Nov blok lahko ustvarimo na več načinov, in sicer s pisanjem kode ali pa s pomočjo namenskih programov za generiranje novih blokov. V našem primeru smo se odločili za uporabo spletnega razvojnega orodja Blockly Developer Tools.

V prvi fazi nadgradnje Ardublocklyja smo za test v Blockly Developer Tools ustvarili in prenesli osnovni blok v programsko okolje Ardublockly. V drugi fazi smo glede na funkcije robota, s katerim želimo nadgraditi naš program, preuredili izgled in funkcijo testnega bloka.

### 6.2.1. Definicija bloka

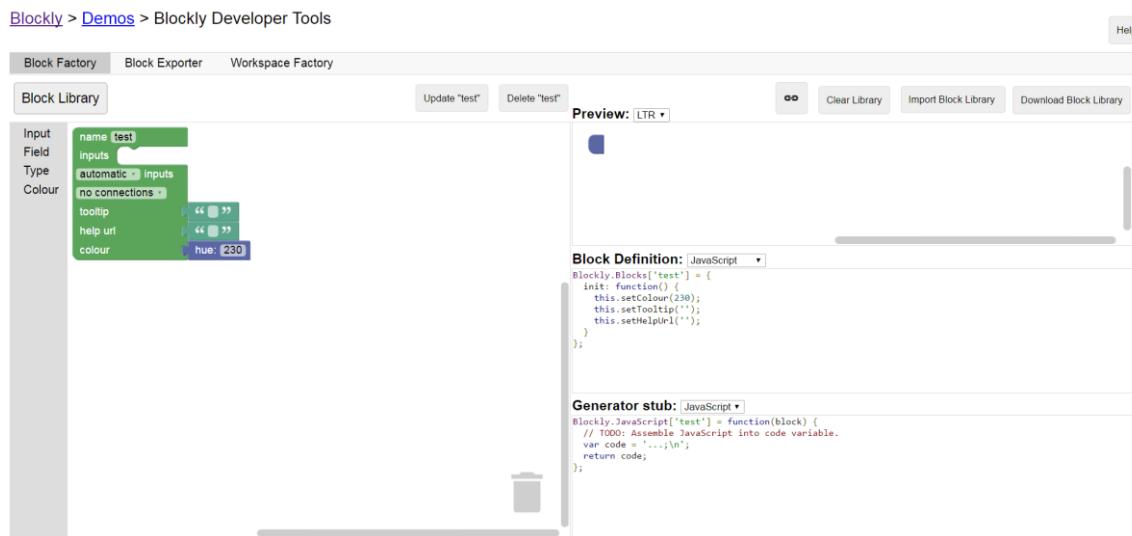
Pod definicijo bloka spada določevanje oblike, vhodnih polj in točk povezave bloka z drugimi bloki. Kot že povedano, bomo definicijo bloka izvedli s pomočjo orodja Blockly Developer Tools, ki temelji na vizualnem načinu programiranja. Z omenjenim razvojnim orodjem lahko gradimo nove bloke, liste orodij in urejamo novonastale bloke v posamezni kategoriji, in sicer z uporabo že definiranih blokov. Postopek uporabe razvojnega orodja Blockly Developer Tools sestoji iz treh delov:

- ustvarjanje novega bloka z uporabo zavihka Block factory in Block exporter,
- ustvarjanje liste orodij z uporabo zavihka Workspace factory in
- urejanje delovnega prostora v posamezni kategoriji v zavihku Workspace factory.

Ker je naš namen uporabe omenjenega razvojnega okolja samo generacija kode, bomo uporabili zavihka Block factory in Block exporter, pri čemer Block factory generira kodo

definicije bloka in generirajočo kodo bloka, Block exporter pa omogoča izvoz kode v JavaScript obliki.

Za definicijo našega testnega bloka imamo v zavihu Block factory na voljo štiri kategorije, in sicer kategorije vhoda, polja, tipa in barve. Ustvarjen nov blok vedno izhaja iz osnovnega bloka, ki je prikazan na sliki 30 v zeleni barvi. V osnovni blok zelene barve najprej vpišemo v polje name ime novega bloka in ga tako poimenujemo. Ime novega bloka je zelo pomembno, saj na ta način ločimo bloke med seboj. Nato po potrebi dodajamo osnovnemu bloku že definirane bloke iz posameznih kategorij in tako tvorimo nov blok. Za naš primer smo uporabili le osnovno obliko, ki je prikazana na sliki 30.



Slika 30: Prikaz zavihka Blockly Factory [40].

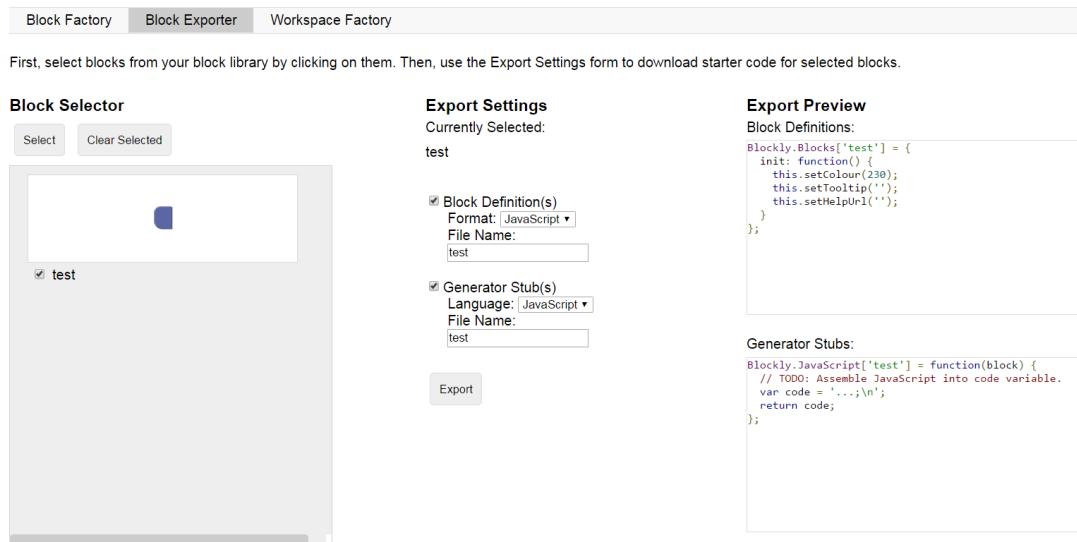
## 6.2.2. Generiranje kode bloka

Pri uporabi blokov se morajo le-ti pretvoriti v kodo za izvedbo njihovih funkcij, ki so predpisane za posamezen blok. Preko uporabe zavihka Block Factory dobimo obliko kode, prilagojeno definiciji bloka, pri čemer se upoštevajo le vhodni parametri. Ostale računske in logične operacije, ki predstavljajo »srce« bloka, moramo določiti sami.

## 6.2.3. Umestitev kode

Po končani definiciji in generaciji kode blok shranimo in nato z uporabo zavihka Block Exporter prenesemo izvorno kodo našega testnega bloka na računalnik. Izgled izvorne kode za izvoz je prikazan na desni strani slike 31. Datoteki definicije in kode bloka moramo najprej poimenovati, nato sledi prenos datoteke s tipko Export.

[Blockly](#) > [Demos](#) > Blockly Developer Tools



Slika 31: Generirana koda za izvoz testnega bloka [40].

Preneseni datoteki z imenom test moramo vstaviti v točno določeno datoteko, iz katere program Ardublockly bere njihovo kodo. Datoteko s kodo definicije bloka vstavimo na lokacijo \blockly\blocks\arduino, generirajočo kodo pa na lokacijo \blockly\generators\arduino.

Da se novi blok lahko prikaže v programu, je potrebno dodati v datoteko index.html stavek <script...> z lokacijo datotek definicije kode (slika 32) in generirajoče kode (slika 33).

```
<!-- Ardublockly - These three files contain the compress version -->
<!-- script src="../blockly/blockly_compressed.js"></script>
<script src="../blockly/blocks_compressed.js"></script>
<script src="../blockly/arduino_compressed.js"></script>-->
<!-- To use the uncompressed version comment out the above and comment in the ones below-->
<script src="../blockly/blockly_uncompressed.js"></script>
<script src="../blockly/blocks/logic.js"></script>
<script src="../blockly/blocks/loops.js"></script>
<script src="../blockly/blocks/math.js"></script>
<script src="../blockly/blocks/text.js"></script>
<script src="../blockly/blocks/lists.js"></script>
<script src="../blockly/blocks/arduino/test.js"></script>
<script src="../blockly/blocks/colour.js"></script>
<script src="../blockly/blocks/variables.js"></script>
<script src="../blockly/blocks/procedures.js"></script>
<script src="../blockly/blocks/arduino/io.js"></script>
<script src="../blockly/blocks/arduino/map.js"></script>
```

Slika 32: Vstavljen script stavek z lokacijo datoteke s kodo definicije bloka.

```

<script src="../blockly/generators/arduino/boards.js"></script>
<script src="../blockly/generators/arduino/io.js"></script>
<script src="../blockly/generators/arduino/lists.js"></script>
<script src="../blockly/generators/arduino/logic.js"></script>
<script src="../blockly/generators/arduino/loops.js"></script>
<script src="../blockly/generators/arduino/map.js"></script>
<script src="../blockly/generators/arduino/math.js"></script>
<script src="../blockly/generators/arduino/test.js"></script>
<script src="../blockly/generators/arduino/procedures.js"></script>
<script src="../blockly/generators/arduino/serial.js"></script>
<script src="../blockly/generators/arduino/servo.js"></script>

```

Slika 33: Vstavljen script stavek z lokacijo datoteke z generirajoče kodo.

Za določitev kategorije, v kateri se bo naš testni blok pojavil, je potrebno dodati v kodo datoteke z imenom ardublockly\_toolbox.js še vrstico, prikazano na sliki 34.

```

Ardublockly.TOOLBOX_XML =
'<xml>' +
'  <sep></sep>' +
'  <category id="robot" name="Robot">' +
'    <block type="test"></block>' +
'  </category>' +
'  <sep></sep>' +

```

Slika 34:Vstavljeni vrstici za naš testni blok.

Rezultat vseh opisanih sprememb je dodan nov testni blok, ki ga je mogoče videti na sliki 35 kot moder pravokotnik oziroma blok v kategoriji Robot. Na ta način smo se prepričali, da je mogoče vstaviti poljuben blok v Ardublockly okolje in s tem smo potrdili možnost spremembe našega izbranega programa. V naslednji fazi bomo obstoječi testni blok uporabili kot osnovo za predelavo s funkcijami mobilnega robota.



Slika 35: Prikaz novo ustvarjenega bloka v programu Ardublockly.

### 6.2.4. Blok Pelji

Za programiranje mobilnega robota smo kodo testnega bloka prilagodili izbrani funkciji pelji. Program pelji predstavlja osnovno funkcijo gibanja mobilnega robota, ki vključuje parametra hitrosti in rotacije. Njuna sprememba določuje način gibanja robota (premočrno gibanje in zavoji). Ker funkcija pelji vsebuje dva parametra, moramo v kodi določiti dve vhodni spremenljivki hitrosti in rotacije, katerih vrednosti se uporabijo v kodi Robot.cpp in kodi knjižnice Robot.h. Testno datoteko z generirajočo kodo na lokaciji \blockly\generators\arduino preimenujemo v pelji in ji spremenimo kodo v končno obliko, kot je prikazana na spodnji sliki 36.

```
Blockly.Arduino['pelji'] = function(block) {
    var speed = Blockly.Arduino.valueToCode(
        block, 'ROBOT_SPEED', Blockly.Arduino.ORDER_ATOMIC) || '100';
    var rotation = Blockly.Arduino.valueToCode(
        block, 'ROBOT_ROTATION', Blockly.Arduino.ORDER_ATOMIC) || '0';
    Blockly.Arduino.addInclude('robot', '#include <Robot.h>');
    Blockly.Arduino.addDeclaration('robot', 'Robot robot;');
    var code = 'robot.pelji(' + speed + ', ' + rotation + ');\n';
    return code;
};
```

Slika 36: Generirajoča koda funkcije pelji.

Ker smo spremenili ime testnega bloka generirajoče kode v pelji, moramo spremeniti tudi definicijo bloka, kot je prikazano na sliki 37.

```
Blockly.Blocks['pelji'] = {
    init: function() {
        this.setColour(230);
        this.setTooltip('');
        this.setHelpUrl('');
        this.setInputsInline(false);
        this.appendDummyInput()
            .appendField("Hitrost: ");
        this.appendValueInput('ROBOT_SPEED')
            .setCheck(Blockly.Types.NUMBER.checkList)
            // .appendField("ROBOT_SPEED");
        this.appendDummyInput()
            .appendField(" Rotacija: ");
        this.appendValueInput('ROBOT_ROTATION')
            .setCheck(Blockly.Types.NUMBER.checkList)
            // .appendField("ROBOT_ROTATION");
        this.setInputsInline(true);
        this.setPreviousStatement(true, null);
        this.setNextStatement(true, null);
    }
};
```

Slika 37: Prikaz kode definicije bloka pelji.

Ko smo preimenovali datoteke in programsko kodo našega testnega bloka, je potrebno še dodati oziroma spremeniti ime testnega bloka v datoteki z imenom ardublockly\_toolbox.js, ki se nahaja na lokaciji \ardublockly\ardublockly. Sprememba je prikazana na sliki

```
Ardublockly.TOOLBOX_XML =
'<xml>' +
'  <sep></sep>' +
'  <category id="robot" name="Robot">' +
'    <block type="pelji"></block>' +
'  </category>' +
'  <sep></sep>' +
```

Slika 38: Sprememba imena bloka.

Potrebno je spremeniti še ime in lokacijo datotek za naš novi blok pelji v datoteki z imenom index.html, ki se nahaja na lokaciji \ardublockly\ardublockly. Spremembe so prikazane na slikah 39 in 40.

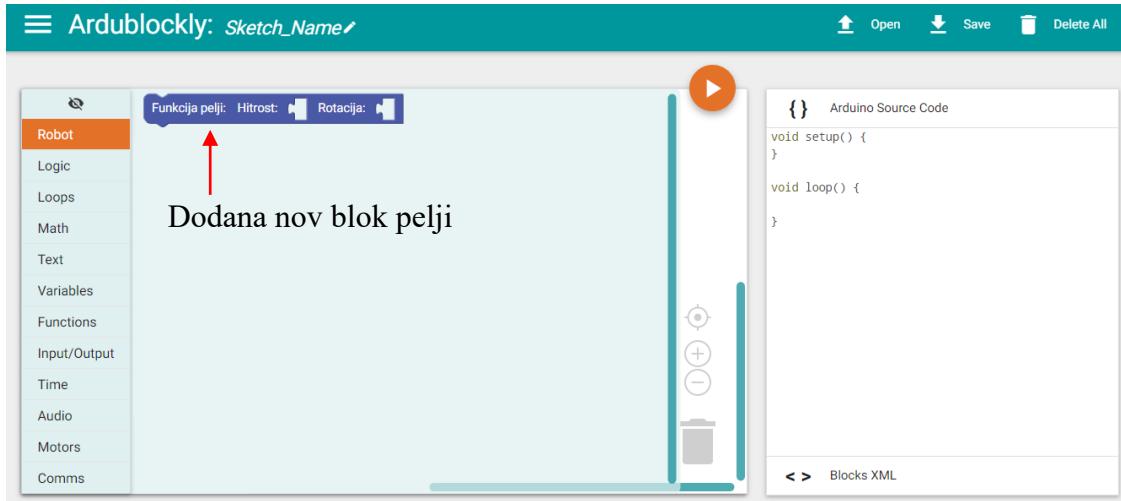
```
<!-- Ardublockly - These three files contain the compress version -->
<!-- script src="../blockly/blockly_compressed.js"></script>
<script src="../blockly/blocks_compressed.js"></script>
<script src="../blockly/arduino_compressed.js"></script-->
<!-- To use the uncompressed version comment out the above and comment in the ones below-->
<script src="../blockly/blockly_uncompressed.js"></script>
<script src="../blockly/blocks/logic.js"></script>
<script src="../blockly/blocks/loops.js"></script>
<script src="../blockly/blocks/math.js"></script>
<script src="../blockly/blocks/text.js"></script>
<script src="../blockly/blocks/lists.js"></script>
<script src="../blockly/blocks/colour.js"></script>
<script src="../blockly/blocks/variables.js"></script>
<script src="../blockly/blocks/procedures.js"></script>
<script src="../blockly/blocks/arduino/pelji.js"></script>
<script src="../blockly/blocks/arduino/io.js"></script>
<script src="../blockly/blocks/arduino/map.js"></script>
```

Slika 39: Prikaz lokacije navezave na datoteko definicije bloka pelji.

```
<script src="../blockly/generators/arduino/boards.js"></script>
<script src="../blockly/generators/arduino/io.js"></script>
<script src="../blockly/generators/arduino/lists.js"></script>
<script src="../blockly/generators/arduino/logic.js"></script>
<script src="../blockly/generators/arduino/loops.js"></script>
<script src="../blockly/generators/arduino/map.js"></script>
<script src="../blockly/generators/arduino/math.js"></script>
<script src="../blockly/generators/arduino/pelji.js"></script>
<script src="../blockly/generators/arduino/procedures.js"></script>
<script src="../blockly/generators/arduino/serial.js"></script>
<script src="../blockly/generators/arduino/servo.js"></script>
```

Slika 40: Prikaz lokacije navezave na datoteko generirajoče kode.

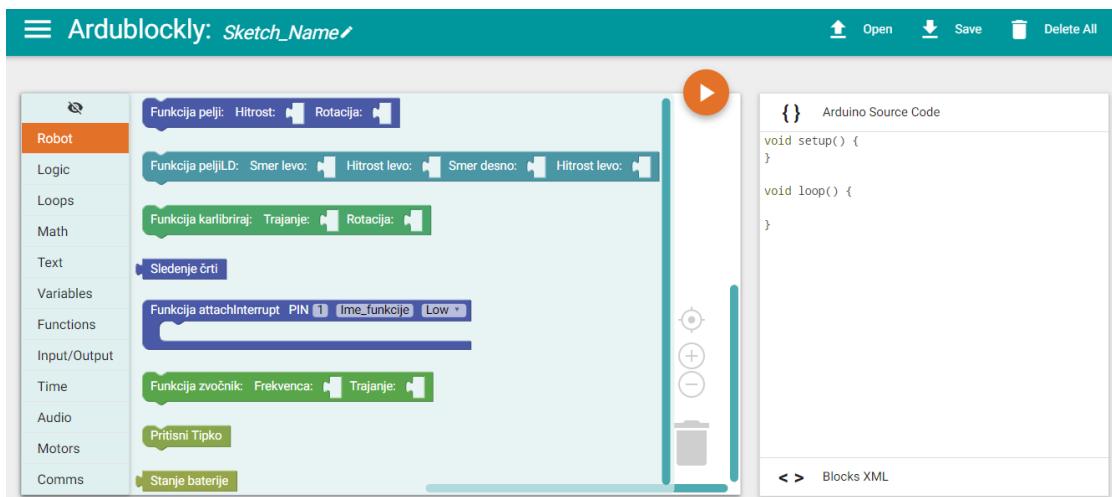
Po končanih opisanih spremembah je naš končni rezultat blok pelji, v kategoriji Robot, kar prikazuje slika 41.



Slika 41: Prikaz spremembe našega testnega bloka v blok pelji.

### 6.3. Rezultati modifikacije in opis novih funkcijskih blokov

V poglavju 6.2.4 Blok Pelji smo prikazali celoten postopek ustvarjanja novega bloka, in sicer od trenutka kreiranja kode bloka pa vse do njegove končne oblike v programu. Na ta način smo ustvarili tudi vse ostale funkcije za programiranje našega mobilnega robota. Končni rezultat modifikacije našega programskega okolja je prikazan na sliki 42.



Slika 42: Končni izgled našega programskega okolja z vstavljenimi novimi bloki.

Na sliki 42 je mogoče opaziti bloke z različnimi oblikami in vnosnimi polji, kar je odvisno od funkcije oz. namena posameznega bloka. Nekateri bloki tako potrebujejo za svojo izvršitev vhodne podatke, čemur služijo prazna polja v obliki koščka sestavljanke. Drugi bloki so samostojni in zato ne potrebujejo nobenega vhodnega parametra ter se samo »pripnejo« obstoječi kodi. Glede na način pripenjanja lahko razdelimo bloke v dve skupini: prva skupina se pripenja v vertikalni smeri in predstavlja v tekstovni kodi skupino vrstic kode, ki ne vračajo nobenih vrednosti, druga skupina blokov, ki se pripenja v horizontalni smeri z obliko koščka sestavljanke, pa vrača vrednosti.

Tako smo preko modifikacije nadgradnje programskega vizualnega okolja ustvarili naslednje bloke z njihovim opisom:

*void pelji(int hitrost, int rotacija)*

Funkcija pelji omogoča robotu gibanje in sprejema vhodne spremenljivke hitrosti in rotacije, na podlagi katerih se izračuna hitrost vrtenja posameznega kolesa (levo in desno). Na ta način se določi hitrost in smer gibanja robota.

*void peljiLD(int smerLevo, int hitrostLevo, int smerDesno, int hitrostDesno);*

Funkcija peljiLD omogoča, da lahko uporabnik določi poljubno pot robota. Funkcija sprejema vhodne spremenljivke hitrosti in smeri. Hitrosti nastavljamo med 0 in 255, smeri pa 0 za vrtenje kolesa nazaj in 1 za vrtenje kolesa naprej. Na ta način se določi hitrost in smer gibanja robota.

*void kalibrirajCrtto(int trajanje)*

Funkcija kalibriraj črto da robotu ukaz za njegovo rotacijo, pri čemer robot meri stanje na senzorjih in si zapomni njihove minimalne in maksimalne vrednosti. Te vrednosti se nato skalirajo in uteženo povprečijo za določevanje pozicije črte.

*int crta()*

Funkcija crta nam uteženo povpreči skalirane vrednosti senzorjev. Skaliranje vrednosti poteka tako, da je na senzorju vrednost 0, če ni zaznana črta, in 1000, če je senzor čisto na črti. Skrajne povprečne vrednosti za 8 senzorjev so -3500 in 3500, ko je robot čisto na levem oz. čisto na desnem senzorju.

*void cakajTipko()*

Funkcija čakaj tipko preverja vsakih 20ms, če je bila tipka pritisnjena. V primeru, da je bila pritisnjena tipka, se program izvede dalje.

*int baterija()*

Funkcija baterija nam poda informacijo o trenutni napetosti baterije, kar uporabniku daje informacijo o trenutnem energijskem stanju baterije.

*void zvocnik(int frekvenca, int trajanje)*

Funkcijo zvočnik uporabimo, kadar želimo, da robot po končani nalogi oz. ob določenem pogoju odda zvočni signal.

*attachInterrupt (digitalPinToInterrupt (pin),ISR,mode)*

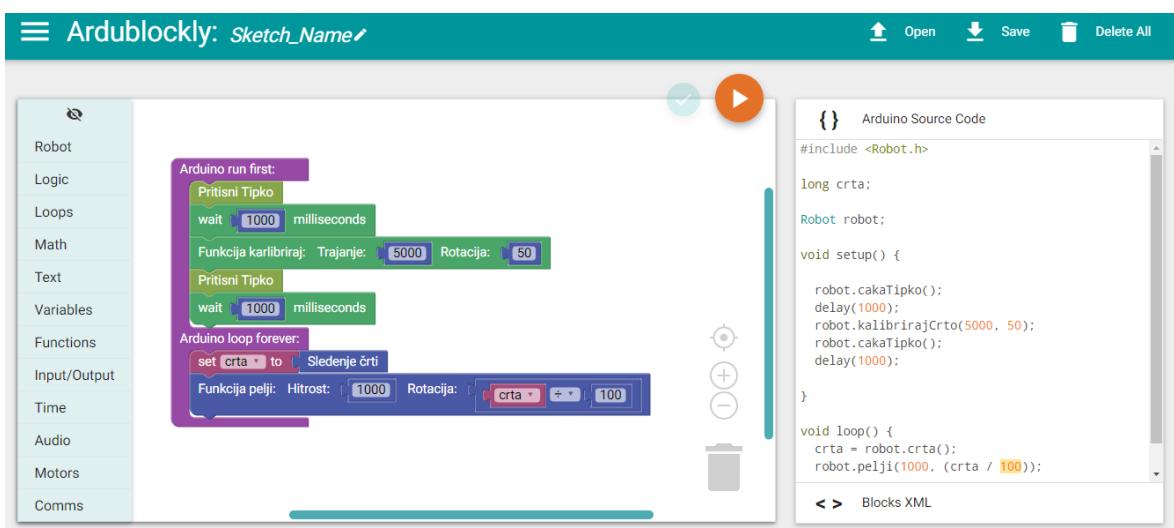
Funkcija se uporablja z namenom izvedbe prekinitev izvajanja trenutnega programa. Tako se pri prekinitvi izvede prekinitvena rutina, s kratico imenovana ISR funkcija, ki jo moramo kot vhodni parameter določiti v funkciji attachInterrupt. Funkcija attachInterrupt nam kot prvi parameter sprejme funkcijo digitalPinToInterrupt (pin), s katero določimo številko pina, ki ga želimo uporabiti. Glede na tip Arduino ploščice so za izvedbo prekinitev točno določene številke pinov, ki to omogočajo. Tako sta za Arduino ploščice Uno, Nano in Mini določena pina številka 2 in 3. Drugi parameter definira ISR (ang. *Interrupt Service Routine*) funkcijo, ki se izvede v trenutku prekinitve programa, zadnji parameter, Mode, pa nam definira trenutek izvedbe prekinitve. Prekinitev se izvede glede na štiri predpisane omejitve (LOW, CHANGE, RISING, FALLING).

## 7. Demonstracija na realnem robotu

Končni rezultat modifikacije programskega okolja so bile dodane naslednje funkcije: pelji, pelji levo in desno, kalibracija, črta, attachinterrupt, zvočnik, pritisni tipko ter baterija. Vse naštete funkcije smo s preizkusom na realnem mobilnem robotu preverili in se tako prepričali o njihovem pravilnem delovanju.

Naš test je zajemal dva dela, prvi v programskem okolju Ardublockly in drugi del na testni stezi. V programskem okolju Ardublockly smo sestavili program s pomočjo novih blokov, ki smo jih ustvarili za potrebo krmiljenja mobilnega robota. V drugem delu smo na mobilni robot prenesli že ustvarjen program in preizkusili njegovo delovanje na testni stezi.

Izvedena sta bila dva testa, da bi testirali vpliv vhodnih parametrov na gibanje robota pri vožnji po testni stezi (črti).



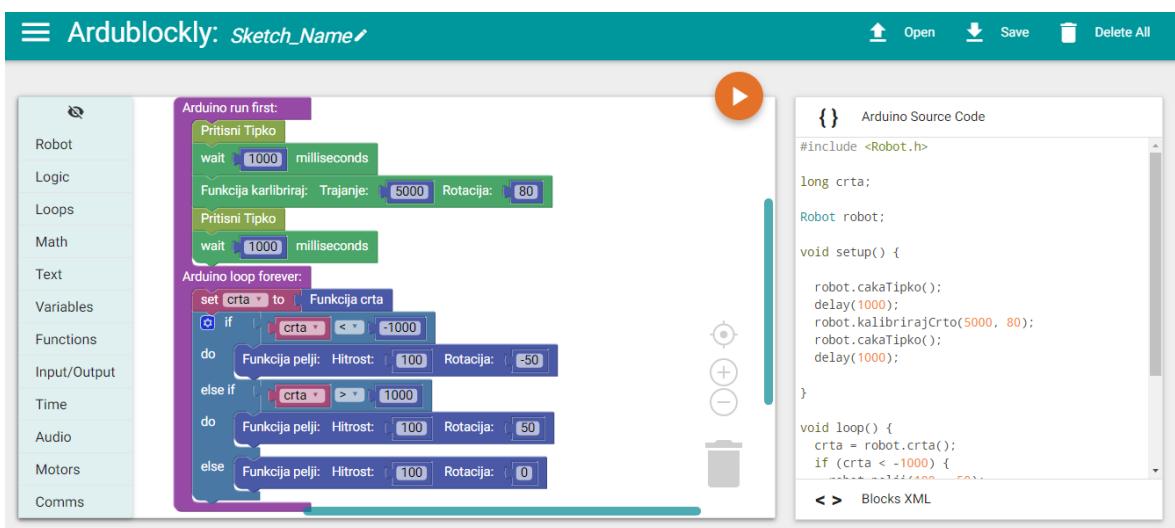
Slika 43: Prikaz prvega testnega programa.

V prvem programu smo ustvarili program za sledenje črte. Ker mobilni robot ob zagonu potrebuje kalibracijo svoje pozicije glede na pozicijo črte, smo program razdelili na dve aktivnosti. Prva aktivnost zajema kalibracijo robota, druga aktivnost pa sledenje črti. Uporabili smo Arduino blok, ki omogoča vnos enkratnih in ponavljajočih aktivnosti. Ker je

kalibracija črte enkratna aktivnost, smo program vpisali v prvi del bloka. Kot je razvidno slike 43, se postopek kalibracije začne s pritiskom na tipko, ki zažene program.

Nato sledi sekunda premora in takoj za tem kalibracija robota, pri čemer se robot vrti v krogu in išče svojo pozicijo glede na črto. Ko se postopek kalibracije zaključi, je spet potreben pritisk tipke. Po sekundi premora preidemo v aktivnost sledenje črti. Sledenje črti je ponavljajoča se aktivnost, zato smo program sledenja črti sestavili v drugem delu bloka. Program sledenja črti vsebuje določitev spremenljivke črta funkciji imenovani črta. Nato sledi funkcija pelji, ki sprejema parametra hitrosti in rotacije. V parameter hitrost nastavimo željeno hitrost robota, kar pomeni, da se bo robot peljal naravnost s konstantno hitrostjo. Ker želimo, da robot sledi črti se mora parameter rotacija spremenijati glede na smer črte. Zato smo v parameter rotacija vstavili spremenljivko črta, katere vrednost se spreminja, saj je spremenljivka črta vezana na funkcijo črta. Parameter rotacije smo delili s faktorjem 100, kar zmanjša hitrost sprememb zasukov koles.

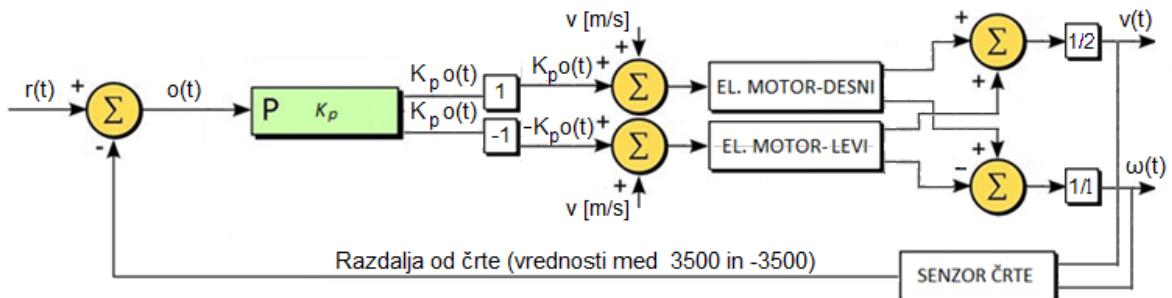
Z namenom izboljšanja programa sledenja črti smo izvedeli še dodatni program. Aktivnost kalibracije robota smo obdržali iz prejšnjega primera. Spremenili smo program sledenja črti z uporabo if stavka in funkcijami pelji. V primeru, da dobimo vrednost iz senzorjev črte manjšo od -1000, se izvrši rotacija robota v eno smer. V primeru, da dobimo signal iz senzorjev črte večji od 1000, pa se izvrši rotacija robota v drugo smer. V primeru, da vrednost iz senzorjev črte ni manjša od -1000 in večja od 1000, se robot pelje naravnost.



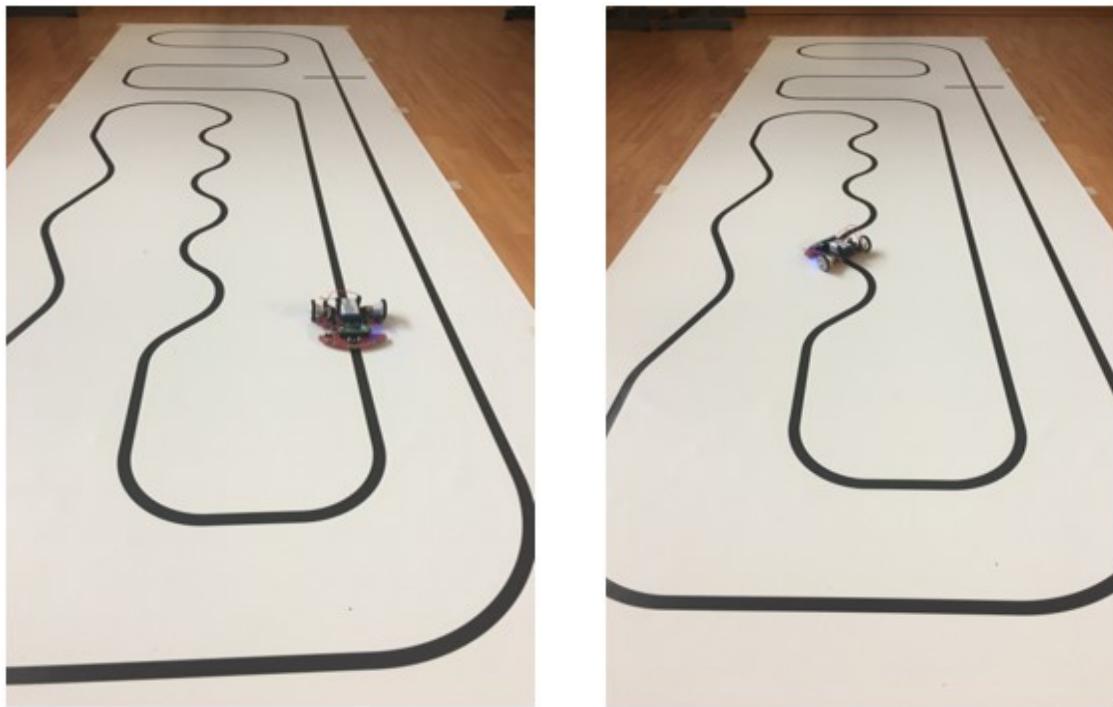
Slika 44: Prikaz drugega testnega programa.

Po končanem programiranju v programskega okolju Ardublockly smo s pomočjo gumba »prenos programske kode na Arduino« prenesli ustvarjen program na mobilni robot. Nato smo preizkusili delovanje novo ustvarjenega programa na testni stezi. Glede na naše začetne vrednosti je bilo mogoče opaziti, da prihaja do prenihajev gibanja robota pri sledenju črte, zato je bilo potrebno vhodne parametre nekoliko prilagoditi. Razlog prenihajev je proporcionalni način krmiljenja mobilnega robota in prevelik koeficient proporcionalnega ojačanja  $K_p$ , ki se s takojšnjim odzivom sistema odzove na napako. V našem primeru je sledenje črti izvedeno s pomočjo senzorjev črte, ki dajejo signal krmilniku, ta pa poslje signal na ustrezen elektromotor (levi ali desni). Signal na posamezen elektromotor je proporcionalen koeficientu proporcionalnega ojačanja  $K_p$  in napaki  $o(t)$ , ta pa je enaka razlike med referenčnim položajem na črti  $r(t)$ , ki je v našem primeru enak nič, in dejanskim

položajem na črti  $i(t)$ . Posameznemu motorju se tako poveča ali zmanjša hitrost vrtenja, s čimer dosežemo zasuk robota v smeri sredine črte. Slika 45 prikazuje blokovni diagram mobilnega robota za sledenje črte, slika 46 pa prikazuje mobilni robot med sledenjem črte na testni stezi.



Slika 45: Poenostavljen blokovni diagram mobilnega robota.



Slika 46: Mobilni robot na testni stezi z novo ustvarjenim programom.

S preizkusom smo dokazali uspešno delovanje ustvarjenega programa z uporabo novih blokov, s čimer smo potrdili tudi pravilno nadgradnjo izbranega programskega okolja. Spoznali smo delovanje mobilnega robota kot celote in njegovega načina gibanja. Opazili smo, da imajo na gibanje robota velik vpliv vhodni parametri, s katerimi lahko dosežemo bolj ali manj sunkovite gibe mobilnega robota pri sledenju črte.



## 8. Zaključek

Na trgu se pojavljajo številni proizvajalci programljivih robotskeh kompletov, ki lahko služijo kot igrače za izobraževalne namene otrok ter ostalih navdušencev nad robotiko ali pa kot napredno izobraževalno sredstvo za učenje robotike na višjem nivoju. Tako imamo na razpolago različne izdelke znanih proizvajalcev kot sta Lego in Fischertechnik, ter izdelke manj znanih start up podjetij. Vsem učnim kompletom je skupna modularna gradnja z uporabo različnih gradnikov in krmilna enota s pripadajočo programsko opremo. Ker je programska oprema prilagojena začetnikom, le-ta večinoma temelji na uporabi vizualnih programskih okolij. Na ta način lahko uporabniki hitro usvojijo programiranje robota in s tem njegovo uporabo.

Kot izobraževalno sredstvo se omenjeni učni kompleti ne uporabljam samo v šolah, temveč tudi v sklopu delavnic poletnih šol in taborov, katerih namen je izobraževanje in promocija robotike. Organizirana so tudi robotska tekmovanja za osnovnošolce in srednješolce v različnih kategorijah, in sicer tako na državnih kot tudi na mednarodni ravni. Na Fakulteti za strojništvo Univerze v Ljubljani se izvaja poletna šola z imenom Poletna šola strojništva, v sklopu katere poteka delavnica na temo mobilnega robota. Učenci se tako učijo in spoznavajo delovanje mobilnega robota, kar obsega tudi programiranje. Tako se nekateri prvič srečajo s programiranjem.

Ker je za neizkušenega uporabnika najlažje programiranje v vizualnem programskem okolju, smo preverili stanje nabora vizualnih programskih okolij za programiranje Arduino platforme. Na podlagi določenih kriterijev smo izvedli selekcijo in izbor končnega programskega okolja za nadgradnjo.

Izbrali smo programsko okolje Ardublockly, v katerega smo dodali vse potrebne funkcije za programiranje mobilnega robota Robošolar, ki se uporablja kot učno sredstvo v Poletni šoli strojništva na Fakulteti za strojništvo Univerze v Ljubljani. Nadgradnja je potekala v več korakih, ki so obsegali aktivnosti od faze umestitve nove kategorije z imenom Robot v orodjarno programa pa vse do faze umestitve novih funkcijskih blokov za programiranje mobilnega robota. Tako smo prišli do naslednjih ugotovitev in rezultatov našega dela:

- 1) Obstaja veliko proizvajalcev robotskih učnih kompletov, ki omogočajo programiranje s priloženo programsko opremo. V osnovnih in srednjih šolah je najpogosteje uporabljen učni komplet LEGO Mindstorms.

- 2) Za izobraževalne namene in promocijo robotike se v Sloveniji izvajajo robotska tekmovanja in poletne šole. Med najbolj uveljavljenimi tekmovanji v Sloveniji sta tekmovanji RoboT in RoboLiga.
- 3) Spoznali smo, da obstaja več različnih delitev in tipov vizualnih programskih jezikov ter okolij.
- 4) Navedli smo prednosti in slabosti uporabe vizualnih programskih jezikov ter njihova področja uporabe.
- 5) Izvedli smo nadgradnjo vizualnega programskega okolja Ardublockly.
- 6) Spoznali smo osnove programiranja v programskem jeziku C++.
- 7) Ustvarili smo nove bloke za programiranje mobilnega robota.
- 8) S testom programa na mobilnem robottu smo dokazali pravilno delovanje novo ustvarjenih blokov in uspešno nadgradnjo vizualnega programskega okolja.
- 9) Spoznali smo vpliv vhodnih parametrov na odzivnost mobilnega robota.

### **Predlogi za nadaljnje delo**

Nadaljnje bi lahko izboljšali videz vmesnika oz. izgled posameznih blokov z nadgradnjo slik v posamezen blok, kar bi začetnikom olajšalo predstavo funkcije bloka. Ker vemo, da je obnašanje robota pri sledenju poti odvisno od vnesenih vrednosti, bi lahko vnosna polja vsebovala že začetne vrednosti. Tako se uporabniku ne bi bilo potrebno pri prvem zagonu programa ukvarjati z iskanjem primernih vrednosti, temveč bi v trenutku zaključka programiranja samo izvedel zagon programa. Za bolj kompleksne naloge robota bi bilo potrebno izdelati dodatne nove bloke, medtem ko bi bloke, ki se zelo pogosto uporabljajo, lahko združili v že sestavljenou skupino blokov kot nekakšno predlogo. Primer vsakokratne aktivnosti robota ob njegovem zagonu je kalibracija črte.

## 9. Literatura

- [1] Lotrič, U. in Žabkar, J. LEGO Mindstorms EV3 [online]. Ljubljana: Fakulteta za računalništvo in informatiko, 2015, str.: 3, 12, 14-16 [ogled april 2017]. Dostopno na: [https://ucilnica.fri.uni-lj.si/pluginfile.php/21015/mod\\_resource/content/2/prosojnicaEV3.pdf](https://ucilnica.fri.uni-lj.si/pluginfile.php/21015/mod_resource/content/2/prosojnicaEV3.pdf)
- [2] Bizaj, E. Magistrsko delo; Možnosti za uporabo modula lego NXT na različnih nivojih izobraževanja [online]. 2013, str. 9 [datum ogleda maj 2017]. Dostopno na: <http://www.ung.si/~library/magisterij/PTF/30Bizaj.pdf>
- [3] Modrobotics.[online]. 2012 [ogled april 2017]. Dostopno na: <http://www.modrobotics.com/cubelets/cubelets-getting-started/>
- [4] Modrobotics.[online]. 2012 [ogled april 2017]. Dostopno na: <https://www.modrobotics.com/2016/10/28/cubelets-blockly/>
- [5] Myatoms. Individual ATOMS [online]. 2017 [ogled april]. Dostopno na: <http://myatoms.com/your-atoms/individual-atoms/>
- [6] Slashgear. Atoms modular robotic building toy kits launch [online]. 2017 [ogled april 2017]. Dostopno na: <https://www.slashgear.com/atoms-modular-robotic-building-toy-kits-launch-14305393/>
- [7] Fischertechnik. Prodajni program [online]. [ogled april 2017]. Dostopno na: <http://www.fischertechnik.si/prog.php>
- [8] Fischertechnik. Kompleti COMPUTING [online]. [ogled april 2017]. Dostopno na: <http://www.fischertechnik.si/prog.php?skup=70-05>
- [9] Procontechnology. [online]. 2017 [ogled april 2017]. Dostopno na: <http://users.tpg.com.au/users/p8king/index.htm>
- [10] Robolink. How to Use Rokit Smart [online]. 2017 [ogled april 2017]. Dostopno na: <http://www.robolink.com/how-to-use-rokit-smart/>
- [11] Robolink. Rokit-smart [online]. 2017 [ogled april 2017]. Dostopno na: <http://www.robolink.com/rokit-smart/>
- [12] Makewonder. Robots, Apps [online]. [ogled april 2017]. Dostopno na: <https://www.makewonder.com/>

- [13] Teacherswithappshttp. Dash and Dot [online]. 2014, [ogled junij 2017] . Dostopno na://www.teacherswithapps.com/dash-and-dot/
- [14] Uran, S. in Pogorelc, J. Robotska tekmovanja ROBObum in RoboT. V: J. Posvet o poučevanju tehnike [online]. 2012, str. 92-108 [ogled junij 2017], Dostopno na: http://www.sazu.si/uploads/files/57dfbe71e126b1a75cebe90f/447-34-3.pdf
- [15] Bošnjak, D. Mladi z robot(k)i nad smeti. V: Delo [online]. 2016, [junij 2017]. Dostopno na: http://www.delo.si/znanje/izobrazevanje/mladi-z-robot-k-i-nad-smeti.html
- [16] Rok, V. LAKOS, Mobilni rooti [online]. [ogled maj 2017]. Dostopno na: http://www.lakos.fs.uni-lj.si/index.php?option=com\_content&view=article&catid=35:lani-laboratorijsa&id=74:rok-vrabi
- [17] FE UNI-LJ. Poletni tabor inovativnih tehnologij [online]. [ogled maj 2017]. Dostopno na: http://www.fe.uni-lj.si/dijaki/poletni\_tabor\_inovativnih\_tehnologij/delavnice/
- [18] Wikipedia. Computer programming [online]. 2017 [ogled maj 2017]. Dostopno na: https://en.wikipedia.org/wiki/Computer\_programming#History
- [19] Nierstrasz, O. Visual Programming [online]. str. 5, [ogled junij 2017]. Dostopno na: http://scg.unibe.ch/download/lectures/pl/PL-12VisualProgramming.pdf
- [20] Fahad-cprogramming. Reverse number in c program code [online]. 2012 [ogled april 2017]. Dostopno na: http://fahad-cprogramming.blogspot.si/2013/12/reverse-number-in-c-program-code.html
- [21] Imgur. Pictures fonline]. 2012 [ogled april 2017]. Dostopno na: http://imgur.com/PfJO2
- [22] Robič, B. Osnove programiranja II (Programski jezik C) [online]. str. 8-15 [ogled april 2017]. Dostopno na: http://studentski.net/gradivo/ulj\_fri\_ri3\_pr2\_sno\_programski\_jezik\_c\_01?r=1
- [23] Wikipedia. Visual programming language [online]. 2017 [ogled maj 2017]. Dostopno na: https://en.wikipedia.org/wiki/Visual\_programming\_language
- [24] Edison. Educational Robotics Programming Software [online]. [ogled april 2017]. Dostopno na: https://meetedison.com/robot-programming-software/
- [25] Oregon State University. A guided tour of Forms/3 [online]. 2001 [ogled maj 207]. Dostopno na: http://web.engr.oregonstate.edu/~burnett/Forms3/Tour/tour.html
- [26] Pošćić, A. Značaj, domene primjene i budući razvoj vizualnog programiranja. V: FER uni zg [online]. Str. 2, 3 [ogled junij 2017]. Dostopno na: https://www.fer.unizg.hr/\_download/repository/KDI%2C\_Antonio\_Poscic.pdf
- [27] Visual programming languages. V: Bridges To Computing Brooklyn College [online]. str. 1, 9 [ogled april 2017]. Dostopno na: http://bridges.brooklyn.cuny.edu/collegenow/modules/S9\_VisualProgrammingLanguages/Lab\_1.pdf
- [28] Al-Mulhem, M. ICS 539 Introduction & Classification [online]. 2007, str. 25-34 [ogled april 2017], Dostopno na: http://slideplayer.com/slide/8806888/

- [29] Mitov, B. Arduino: The Visuino project. V: Blaise Pascal Magazine [online]. 2015, Nr 3, str. 37-42 [ogled junij 2017]. Dostopno na: [http://www.mitov.com/Blaise\\_41\\_UKTotal\\_37\\_42.pdf](http://www.mitov.com/Blaise_41_UKTotal_37_42.pdf)
- [30] Ahmand, R. Visual languages: A new way of programming. V: Malaysian journal of computer science [online]. 1999, vol. 12, str. 77 [ogled maj 2017]. Dostopno na: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.539.9867&rep=rep1&type=pdf>
- [31] Ray, P. A Survey on Visual Programming Languages in Internet of Things. V: Hindawi Scientific Programming [online]. 2017, str. 5 [ogled junij 2017]. Dostopno na: <https://www.hindawi.com/journals/sp/2017/1231430/abs/>
- [32] NETLab Toolkit. NETLab Toolkit – Tangible Interaction and Media [online]. [ogled maj 2017]. Dostopno na: <http://www.netlabtoolkit.org/flash>
- [33] Sparkfun. Alternative Arduino Interfaces [online]. [ogled maj 2017]. Dostopno na: <https://learn.sparkfun.com/tutorials/alternative-arduino-interfaces/ardublock>
- [34] S4A. About S4A [online]. [ogled april 2017]. Dostopno na: <http://s4a.cat/>
- [35] Modkit blog. A Modkit year [online]. 2010 [ogledano april 2017]. Dostopno na: <http://blog.modk.it/2010/08/modkit-year.html>
- [36] Blogminibloq. Documentation [online]. [ogled april 2017]. Dostopno na: <http://blog.minibloq.org/p/documentation.html>
- [37] Visuino. Home [online]. 2017 [ogled junij 2017]. Dostopno na: <https://www.visuino.com/>
- [38] Google developers. Try Blockly [online]. [ogled april 2017]. Dostopno na: <https://developers.google.com/blockly/>
- [39] Github. Ardublockly [online]. 2017 [ogled april 2017]. Dostopno na: <https://github.com/carlosperate/ardublockly>
- [40] Blockly-demo. Blockly Developer-block factory [online]. [ogled marec 2017]. Dostopno na: <https://blockly-demo.appspot.com/static/demos/blockfactory/index.html>
- [41] Rok, V. LAKOS, Mobilni rooti [online]. [ogled maj 2017]. Dostopno na: <http://www.lakos.fs.uni-lj.si/images/2016/PS/Lakosbot.pdf>
- [42] Wikipedija. Git [online]. 2017 [ogled marec 2017]. Dostopno na: <https://sl.wikipedia.org/wiki/Git>
- [43] Google developers. Custom Blocks [online]. [ogled marec 2017]. Dostopno na: <https://developers.google.com/blockly/guides/create-custom-blocks/overview>



## 10. Priloga 1 – Knjižnica Robot.h

```
/*
Knjiznica za robota, uporabljenega na poletni soli FS 2016.
Ustvaril: rok.vrubic@fs.uni-lj.si, julij-avgust 2016.
V javni domeni.
*/
#ifndef ROBOT_H
#define ROBOT_H

#include "Arduino.h"

// definicije vseh pinov mikrokrnilnika, po shemi
// digitalni
#define ENCLA 2
#define ENCRA 3
#define LON 4
#define DON 4
#define PWML1 5
#define PWML2 6
#define S0 7
#define S1 8
#define PWMR1 9
#define PWMR2 10
#define BUT 11
#define S2 12
#define BUZZ 13
// analogni
#define MUX 0
#define DS1 1
#define DS2 2
#define DS3 3
#define DS4 4
#define VBAT 5
```

```

class Robot {
public:
    // konstruktorji
    Robot();

    // javne funkcije
    void pelji(int hitrost, int rotacija);
    void peljiLD(int smerLevo, int hitrostLevo, int smerDesno, int hitrostDesno);
    // sledenje crti
    void preberiSenCrte();
    void kalibrirajCrto(int trajanje, int rotacija);
    void resetirajKalibracijoCrte();
    int crta();
    // tipka
    void cakajTipko();
    // senzorji razdalje
    void razdalja();
    // baterija
    int baterija();
    // buzzer
    void zvocnik(int frekvenca, int trajanje);
    // bluetooth
    void zacniKomunikacijo();
    void telemetrija(char *oznaka, int vrednost);
    void telemetrija(char *oznaka, int *vrednost, int dolzina);
    void bluetoothKrmiljenje();

    // javne spremenljivke
    int senCrte[8] = {0, 0, 0, 0, 0, 0, 0, 0};
    int senRazdalje[4] = {0, 0, 0, 0};
    volatile int enkoderja[2] = {0, 0};
    // kalibrirane vrednosti min max
    int _senCrteMin[8] = {1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023};
    int _senCrteMax[8] = {0, 0, 0, 0, 0, 0, 0, 0};
    // kalibrirane vrednosti
    int _senCrteKalibrirano[8] = {0, 0, 0, 0, 0, 0, 0, 0};

private:
    // privatne funkcije
    void _prizgiSen();
    void _ugasniSen();
    void _preberiKalibriranoCrto();

    // privatne spremenljivke
    // zadnji podatek crte
    int _zadnjaCrta = 0;
};#endif

```

## 11. Priloga 2 – knjižnica Robot.cpp

```
/*
Knjiznica za robota, uporabljenega na poletni soli FS 2016.
Ustvaril: rok.vrubic@fs.uni-lj.si, julij-avgust 2016.
V javni domeni.
*/
#include "Robot.h"
#include "Arduino.h"

// konstruktor - inicializacija
Robot::Robot() {
    // definicije pinov, vhodi ali izhodi
    for (int i = 2; i <= 13; i++) {
        if (i == ENCLA || i == ENCRA || i == BUT)
            pinMode(i, INPUT);
        else
            pinMode(i, OUTPUT);
    }

    // zacetno stanje pinov - hitrost = 0, rotacija = 0
    pelji(0, 0);
}

/*
funkcija za premik robota
hitrost - transverzalna hitrost robota
rotacija - hitrost rotacije robota (neg. - protiurno, poz. - sourno)
primeri:
    robot.pelji(0, 0); // robot na miru
    robot.pelji(75, 0); // robot naravnost s hitrostjo 75
    robot.pelji(0, -50); // vrtenje v levo na mestu
    robot.pelji(100, 50); // zavijanje desno ob pomiku naprej
*/
void Robot::pelji(int hitrost, int rotacija) {
```

```

// določitev hitrosti posameznih koles
int levo = hitrost + rotacija;
int desno = hitrost - rotacija;
// omejitev vrednosti
if (levo > 255)
    levo = 255;
else if (levo < -255)
    levo = -255;
if (desno > 255)
    desno = 255;
else if (desno < -255)
    desno = -255;
// nastavitev glede na smer (naprej/nazaj)
// levo kolo
if (levo >= 0) {
    analogWrite(PWML2, 0);
    analogWrite(PWML1, levo);
} else {
    analogWrite(PWML1, 0);
    analogWrite(PWML2, -levo);
}
// desno kolo
if (desno >= 0) {
    analogWrite(PWMR2, 0);
    analogWrite(PWMR1, desno);
} else {
    analogWrite(PWMR1, 0);
    analogWrite(PWMR2, -desno);
}
}

void Robot::peljiLD(int smerLevo, int hitrostLevo, int smerDesno, int hitrostDesno) {
    if (smerLevo) {
        analogWrite(PWML2, 0);
        analogWrite(PWML1, hitrostLevo);
    }
    else {
        analogWrite(PWML1, 0);
        analogWrite(PWML2, hitrostLevo);
    }
    if (smerDesno) {
        analogWrite(PWMR2, 0);
        analogWrite(PWMR1, hitrostDesno);
    }
    else {
        analogWrite(PWMR1, 0);
        analogWrite(PWMR2, hitrostDesno);
    }
}

```

```

/*
  prebere vrednosti senzorjev crte
  razlika med prizgano in ugasnjeno vrednostjo + 1023
*/
void Robot::preberiSenCrte() {
    int pini[3] = {S0, S1, S2};
    int onVrednosti[8] = {1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023};
    _prizgiSen();
    for (int i = 0; i < 8; i++) {
        int b0 = bitRead(i, 0);
        int b1 = bitRead(i, 1);
        int b2 = bitRead(i, 2);
        digitalWrite(pini[0], b0);
        digitalWrite(pini[1], b1);
        digitalWrite(pini[2], b2);
        onVrednosti[i] = analogRead(MUX);
    }
    _ugasniSen();
    for (int i = 0; i < 8; i++) {
        int b0 = bitRead(i, 0);
        int b1 = bitRead(i, 1);
        int b2 = bitRead(i, 2);
        digitalWrite(pini[0], b0);
        digitalWrite(pini[1], b1);
        digitalWrite(pini[2], b2);
        senCrte[i] = onVrednosti[i] + 1023 - analogRead(MUX);
    }
}

/*
  kalibrira senzorje za crto, trajanje doloca cas postopka
  maksimalne in minimalne vrednosti napise v polja
*/
void Robot::kalibrirajCrto(int trajanje, int rotacija) {
    pelji(0, rotacija);
    long cas = millis();
    do {
        preberiSenCrte();
        for (int i = 0; i < 8; i++) {
            if (senCrte[i] > _senCrteMax[i])
                _senCrteMax[i] = senCrte[i];
            if ((senCrte[i] < _senCrteMin[i]))
                _senCrteMin[i] = senCrte[i];
        }
    } while (millis() < cas + trajanje);
    pelji(0, 0);
}

```

```

/*
    ponastavi minimalne in maksimalne kalibrirane vrednosti za crto
*/
void Robot::resetirajKalibracijoCrte() {
    for (int i = 0; i < 8; i++) {
        _senCrteMin[i] = 1023;
        _senCrteMax[i] = 0;
    }
}

/*
    utezeno povprecenje kalibriranih vrednosti
    -3500 pomeni crta na levem senzorju
    +3500 pomeni crta na desnem senzorju
    0 pomeni crto na sredini
    enota je priblizno 1/100 mm ->
    3500 pomeni, da je crta 35 mm od sredine desno
*/
int Robot::crta() {
    _preberiKalibriranoCrto();
    long utezenaVsota = 0;
    int vsota = 0;
    int naCrti = 0;
    for (int i = 0; i < 8; i++) {
        if (_senCrteKalibrirano[i] > 500) {
            naCrti = 1;
        }
        if (_senCrteKalibrirano[i] > 250) {
            utezenaVsota += (long)_senCrteKalibrirano[i] * i * 1000L;
            vsota += _senCrteKalibrirano[i];
        }
    }
    _zadnjaCrta = utezenaVsota / vsota - 3500;

    int max = 0;
    int maxi = -1;
    for (int i = 0; i < 8; i++) {
        if (_senCrteKalibrirano[i] > max) {
            max = _senCrteKalibrirano[i];
            maxi = i;
        }
    }
    if (maxi == 0) {
        _zadnjaCrta = -3500;
    }
    else if (maxi == 7) {
        _zadnjaCrta = 3500;
    }
}

```

```

if (!naCrti) {
    if (_zadnjaCrta < 0)
        _zadnjaCrta = -3500;
    else
        _zadnjaCrta = 3500;
}

return (int)_zadnjaCrta;
}

/*
preverja stanje tipke vsakih 20ms
zapiska ob pritisku
*/
void Robot::cakajTipko() {
    while (digitalRead(BUT) == HIGH) {
        delay(20);
    }
    zvocnik(440, 100); // pisk
}

// prebere senzorje razdalje (0 - 1023)
void Robot::razdalja() {
    for (int i = 0; i < 4; i++) {
        senRazdalje[i] = analogRead(i+1);
    }
}

// prebere voltazo baterije, 100 je 10V, 0 je 8V
int Robot::baterija() {
    int bat = analogRead(5);
    bat = (bat - 818) * 100 / 205;
    return bat;
}

// zapiska s frekvenco in trajanjem
void Robot::zvocnik(int frekvenca, int trajanje) {
    long zacetek = micros();
    int t = (int)(500000L / (long)frekvenca);
    while (micros() < zacetek + (long)trajanje * 1000) {
        digitalWrite(BUZZ, HIGH);
        delayMicroseconds(t);
        digitalWrite(BUZZ, LOW);
        delayMicroseconds(t);
    }
}

// vzpostavi bluetooth komunikacijo
void Robot::zacniKomunikacijo() {

```

```

    Serial1.begin(9600);
}

// 3 znaki za oznako
void Robot::telemetrija(char *oznaka, int vrednost) {
    Serial1.print("#"); // zacetek podatka
    Serial1.print(oznaka);
    Serial1.print(":");
    Serial1.print(micros()); // cas
    Serial1.print(":");
    Serial1.println(vrednost);
}

// za polja spremenljivk
void Robot::telemetrija(char *oznaka, int *vrednost, int dolzina) {
    Serial1.print("#"); // zacetek podatka
    Serial1.print(oznaka);
    Serial1.print(":");
    Serial1.print(micros()); // cas
    Serial1.print(":");
    for (int i = 0; i < dolzina - 1; i++) {
        Serial1.print(vrednost[i]);
        Serial1.print(",");
    }
    Serial1.println(vrednost[dolzina-1]);
}

// krmiljenje prek bluetootha z aplikacijo LakosBot
void Robot::bluetoothKrmiljenje() {
    if (Serial1.available()){
        switch(Serial1.read()){
            case('L'):
            {
                while (!Serial1.available()){
                }
                int dirLeft = Serial1.read();
                while (!Serial1.available()){
                }
                int speedLeft = Serial1.read();
                if (speedLeft > 100) {
                    speedLeft = 100;
                }
                if(dirLeft == 1){
                    analogWrite(PWML2, 0);
                    analogWrite(PWML1, speedLeft);
                }
                else {
                    analogWrite(PWML1, 0);
                    analogWrite(PWML2, speedLeft);
                }
            }
        }
    }
}

```

```

        }
        break;
    }
    case('R'):
    {
        while (!Serial1.available()){
        }
        int dirRight = Serial1.read();
        while (!Serial1.available()){
        }
        int speedRight = Serial1.read();
        if (speedRight > 100) {
            speedRight = 100;
        }
        if(dirRight == 1){
            analogWrite(PWMR2, 0);
            analogWrite(PWMR1, speedRight);
        }
        else {
            analogWrite(PWMR1, 0);
            analogWrite(PWMR2, speedRight);
        }
        break;
    }
}
}

void Robot::_przgiSen() {
    digitalWrite(LON, HIGH);
    delayMicroseconds(250);
}

void Robot::_ugasniSen() {
    digitalWrite(LON, LOW);
    delayMicroseconds(250);
}

void Robot::_preberiKalibriranoCrto() {
    preberiSenCrte();
    for (int i = 0; i < 8; i++) {
        _senCrteKalibrirano[i] = (int)((long)(senCrte[i] - _senCrteMin[i]) * 1000L) /
        (_senCrteMax[i] - _senCrteMin[i]));
        if (_senCrteKalibrirano[i] < 0)
            _senCrteKalibrirano[i] = 0;
        else if (_senCrteKalibrirano[i] > 1000)
            _senCrteKalibrirano[i] = 1000;
    }
}

```