Raspberry Pi multimedijsko računalo

Komlinović, Fran

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Fran Komlinović

Raspberry Pi

ZAVRŠNI RAD

Sisak, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Fran Komlinović

Matični broj: 0016122153

Studij: Primjena informacijske tehnologije u poslovanju

Raspberry Pi

ZAVRŠNI RAD

Mentor/Mentorica:

Izv. prof. dr. sc. Igor Balaban

Varaždin, prosinac 2021.

Fran Komlinović

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog rada je Raspberry Pi računalo i njegove mogućnosti. Današnja tehnologija nam dozvoljava da vrlo jednostavno izradimo robote u slobodno vrijeme za malo novca, a Raspberry Pi je jedan od najboljih i najjeftinijih načina za ulazak u svijet robotike i malih računala. Koristeći razne razvojne okvire kao što je Pi4J vrlo je jednostavno upravljati takvim računalom što doprinosi sve bržem razvoju robotike. Cilj ovog rada je napraviti robot koji se upravlja bežičnom tipkovnicom spojenom na Raspberry Pi računalo, na kojem je pokrenuta Java aplikacija koja pretvara ulaz tipkovnice u pokrete elektromotora koji na kraju pokreću robot. U ovom završnom radu zastupljen je svaki segment od ulaza preko tipkovnice do konačnog pokreta robota.

Ključne riječi: Raspberry Pi; Java; Pi4J; elektromotor; robot;

Sadržaj

1. Uvod

Tema ovog rada je korištenje Raspberry Pi računala za kontroliranje malog vozila pokretanog s dva elektromotora. Izrada ovog koncepta uvelike pomaže shvaćanju osnova robotike, programiranja i elektronike što je naša sadašnjost.

Kroz poglavlja upoznati ćemo se sa svim segmentima kontrole robota, počevši od pritiska tipke, bežičnog prijenosa signala do računala, pretvaranja tog signala u računalu razumljivu naredbu, pretvaranja te naredbe u signal na Raspberry Pi računalu, pretvaranje GPIO pin signala u informaciju elektromotoru koji na kraju okreće kotače i pokreće naše vozilo.

Ovaj koncept se također može prenijeti na kontroliranje bilo kojeg elektronskog elementa.

Krajnji cilj ovog rada je potpuno funkcionalno vozilo koje se može kretati u sve strane, a upravljano je tipkovnicom.

Kako je autor i sam projektant i programer informacijskih sustava, s interesom u robotiku, odabir teme bio je potpuno logičan.

2. Robotika

„Robotika je područje znanja i tehnologije koje se bavi pitanjima mehaničke analize, dizajna, upravljanja, mjerenja, primjene i održavanja robota. Roboti se danas široko koriste u znanstvenim istraživanjima, proizvodnoj tehnologiji, građevinarstvu, prometu i poljoprivredi, kao i u medicini, podvodnim istraživanjima i svemirskim istraživanjima. Teorija i tehnologija robota i manipulatora interdisciplinarno je područje studija koje zahtijeva suradnju stručnjaka iz različitih znanstvenih disciplina.“ [1, p. 1]

Robotika se može podijeliti na nekoliko grana:

1. Teoretska robotika
2. Općenita robotika
3. Mjeriteljska robotika
4. Robotika kretanja
5. Robotika za medicinske primjene i rehabilitaciju
6. Industrijska robotika
7. Uslužna robotika
8. Mikro robotika
9. Robotika za neindustrijsku primjenu [1]

2.1. Povijest i razvoj robotike

Prvi spomen robotike kao koncepta dogodio se 1917. godine kada je Joseph Čapek napisao kratku priču Opilec. Riječ robot dolazi od češke riječi robota što znači kmet ili radnik. [2] Evolucija industrijskih robota može se podijeliti u četiri kategorije. Prve tri kategorije su iz vremena od 1950. do 1999. godine. Roboti četvrte generacije su roboti 21. stoljeća, a karakteriziraju ih „inteligentne“ značajke poput sposobnosti izvođenja naprednih izračuna, logičko zaključivanje, dubinsko učenje, složene strategije i suradničko ponašanje. [3]

2.1.1. Prva generacija industrijskih robota

Ova generacija robota počinje od oko 1950. godine i traje do 1967. godine. Roboti tih ranih generacija bili su strojevi koji nisu imali nikakvu komunikaciju sa vanjskim okruženjem nego su svoje zadatke izvršavali bez obzira na bilo koje vanjske faktore. Gotovo svi roboti te generacije koristili su pneumatske aktuatore te mehaničke graničnike. Takvi roboti mogli su izvršavati samo najjednostavnije zadatke. [3]

2.1.2. Druga generacija industrijskih robota

Drugu generaciju robota koja traje od 1968. do 1977. godine obilježavaju elementarne mogućnosti prepoznavanja vanjskog okruženja i osnovne mogućnosti samoprilagodljivog ponašanja. Roboti druge generacije koristili su servo kontrolere koji su im omogućili izvođenje kretanja od točke do točke. Ovi roboti su u odnosu na prvu generaciju mogli izvršavati i složenije zadatke no nisu bili svestrani, odnosno svaki je robot imao svoju programsku podršku posvećenu određenom zadatku. U to vrijeme bilo je gotovo nemoguće da jedan robot radi različite zadatke jer bi to zahtijevalo temeljito reprogramiranje. U toj generaciji pojavljuje se dijagnostika, ali samo ona vezana uz kvarove. Bila je implementirana na način da se upali žaruljica koja signalizira da kvar postoji, bez ikakvih dodatnih informacija o kakvom se kvaru radi i što je uzrok kvara. [3]

2.1.3. Treća generacija industrijskih robota

Od 1978. do kraja tisućljeća (1999. godine) dolazi do razvoja treće generacije industrijskih robota. Njih karakterizira veći stupanj interakcije s operaterom i okolinom. Roboti ove generacije imali su mogućnost samoprogramiranja te su se mogli reprogramirati kako bi izvršavali različite zadatke. Također, roboti su bili povezani sa CAD-om ili bazom podataka. Počeli su se koristiti i senzori koji su davali podatke koji pomažu robotu da prilagodi kretanje obzirom na promjenu okoline. Dijagnostika se također poboljšala, roboti su mogli napraviti i izvještaj o mjestu i vrsti kvara. Ova generacija je počela s uvođenjem osnovne vrste inteligencije. [3]

2.1.4. Suvremeni roboti

Četvrta generacija robota sastoji se od inteligentnih robota koji uključuju napredna računala za rasuđivanje i učenje i sofisticirane senzore koji pomažu kontrolnim jedinicama da se učinkovitije prilagode različitim okolnostima. Pojavile su se i nove tehnologije, razne hardware platforme te SBCs (Single Board Computers) odnosno malena računala poput Raspberry Pi-a koji je i korišten u ovom projektu.

2.2. Komponente robota

Teško je odrediti kada elektronički ili mehanički objekt postaje robot, ali možemo pronaći neke zajedničke komponente koje postoje kod svih robota. Roboti bi se trebali kretati, imati senzore i neki oblik inteligencije. Iz anatomske perspektive, dijelovi robota bili bi: izvor energije, kontrolna jedinica, senzori, akcijski dijelovi i tijelo robota. [4]

2.2.1. Izvor energije

Energija je osnovna komponenta robota jer bez nje robot ne može nikako funkcionirati. Kretanje robota može se izvršavati pomoću pneumatike, hidraulike ili električne energije. Električna energija može se dobiti na više načina, a najčešći su: baterija, generator sa unutarnjim izgaranjem, spajanje na statični izvor struje, solarna energija i drugi. Električna energija neophodna je za rad računala koje kontrolira robot. U slučaju ovog projekta izvor energije za pomicanje kotača su 4 AA baterije snage 1.5 volti što čini ukupnu snago od 6 volti samo za pokretanje elektromotora. Raspberry Pi električnu energiju dobiva iz malog prijenosnog punjača.

2.2.2. Kontrolna jedinica

Kontrolna jedinica služi da bi povezala i organizirala kompletan robot. Može i ne mora sadržavati umjetnu inteligenciju. U ovom slučaju implementirana je putem Raspberry Pi računala te kontrolera elektromotora. Nema nikakvu umjetnu inteligenciju nego služi isključivo pretvaranju ulaza s tipkovnice u pokrete elektromotora.

2.2.3. Akcijski dijelovi

Akcijski dijelovi služe da bi pomaknuli fizičke dijelove uz pomoć pohranjene energije u izvorima. To su najčešće elektromotori, kemikalije, pneumatika, hidraulika ili drugi načini pokretanja fizičkih elemenata. U našem slučaju koriste se 2 DC elektromotora bez četkica koji su spojeni na svaki kotač te na taj način pokreću vozilo.

2.2.4. Senzori

Senzori su uređaji koji registriraju vanjske promijene i pretvaraju ih u signale robotu. Koriste se kako bi robot dobio informaciju o okolini te sukladno s njome korigirao svoje ponašanje. Također, koriste se za različite oblike mjerenja te za dijagnostiku. U slučaju ovoga vozila, senzora nema, a vanjske podražaje prima osoba koja kontrolira robot tipkovnicom.

2.2.5. Kućište

Kućište je dio robota kojemu je svrha na najbolji način fizički povezati sve dijelove. Služi estetici samog robota, ali i čuva robot od oštećenja. Ovo vozilo ima kućište od kartona jer je dovoljno lagano da se može lako kretati, također je na karton najlakše posložiti sve dijelove u smislenu cjelinu.

2.3. Robotika danas

Robotika danas postala je puno pristupačnija nego što je bila prije. Mala računala su relativno jeftina, postoje razni paketi elektromotora, senzora i ostalih dodataka kao što je CamJam Edukit koji je i korišten u ovom projektu. Naglasak na robotiku se stavlja i prilikom edukacije djece u školama i vanškolskim aktivnostima.

3. Električna energija i elektromotori

3.1. Električna energija

Električna energija danas je postala nezaobilazni oblik energije. Teško je zamisliti život bez električne energije. Sve više se koristi i u vozilima pa je tako i ovaj projekt pokretan električnom energijom. U ovom radu biti će objašnjeno kako se generira, skladišti, prenosi i troši. U ovom projektu, električna energija se generira na razne načine, odnosno skladišti se u bateriju preko električne energije kupljene od distributera. Iz tih baterija uzimamo električnu energiju i njome pokrećemo Raspberry Pi i elektromotore. Osnovni elementi električne energije biti će objašnjeni u nastavku.

3.1.1. Napon

Napon je prvi koncept koji treba razumjeti u vezi električne energije. Napon je količina potencijalne energije između dvije točke u strujnom krugu. Jedna točka ima više električnog naboja nego druga, a njihova razlika naziva se naponom. Napon se mjeri u voltima a naziv mjerne jedinice dolazi od prezimena talijanskog fizičara po imenu Alessandro Volta koji je izumio prvu kemijsku bateriju. [5]

3.1.2. Struja

Struja je količina naboja koji teče kroz krug u određenom vremenu. Struja se mjeri u amperima koji je definiran kao kulon po sekundi. Amper je ime dobio po francuskom fizičaru po imenu André-Marie Ampère, jednom od prvih znanstvenika na području elektromagnetizma. [5]

3.1.3. Otpor

Otpor je mjera sprječavanja toka električne struje kod nekog objekta. To znači da će sklop a manjim otporom propustiti više električnog naboja nego sklop s većim otporom. Označava se sa grčkim slovom Ω.

3.2. Baterije i pohrana električne energije

Baterije su način pohrane električne energije. Sastoje se od skupa jedne ili više stanice čije kemijske reakcije stvaraju protok elektrona u strujnom krugu, a sastavljene su od anode, katode i elektrolita. Prilikom spajanja anode i katode u krug, dolazi do kemijske reakcije između

anode i elektrolita. Ta reakcija uzrokuje protok elektrona kroz krug i povratak u katodu gdje se odvija još jedna kemijska reakcija. Kada se materijal u katodi ili anodi potroši, bateriju smatramo praznom. Baterije koje odbacujemo nakon upotrebe zovemo primarne, a one koje se mogu ponovno napuniti nazivamo sekundarne. [6]

3.2.1. Povijest baterije

1791. godine talijanski fizičar Volta je objavio otkrića o stvaranju napona pomoću tkanine natopljene slanom vodom, a 1800. godine je stvorio prvu bateriju pod nazivom The Voltanic Pile. The Voltanic Pile se sastojala od cinkovih i bakrenih ploča odvojenih krpom natopljenom slanom vodom. Najveći problemi te baterije bili su veličina, curenje vode, i kratak životni vijek. Kroz povijest se ta baterija poboljšavala. Prva sekundarna baterija nastala je 1859. godine, a izumio ju je Gaston Plante. Takva baterije stvorena je koristeći dva valjana lima olova potopljena sumpornom kiselinom. Reverziranjem električne struje kroz bateriju kemija bi se vratila u prvobitno stanje. 1881. godine Camille Alphonse Faure umjesto olovnih listova stavio je ploče što je znatno olakšalo proizvodnju baterija. Takva baterija danas se nalazi u automobilima. Elektrolit u tekućem stanju je stvarao veliki problem kod transporta te je zbog tog problema nastala baterija sa suhom ćelijom, odnosno cink-ugljične baterije. Pedesetih godina prošlog stoljeća Lewis Urry, Paul Marsal i Karl Kordesch zamijenili su amonijev klorid alkalnom supstancom. Takve alkalne baterije postale su standard. Prva litij-ionska baterija nastala je 1985., a prva komercijalna takva baterija 1991. godine. [6]

3.2.2. Dijelovi baterije

Kao što je i ranije navedeno, osnovni dijelovi baterije su anoda, katoda i elektrolit. A svi ti dijelovi nalaze se u kućištu.

Anoda je prvi dio koji ćemo spominjati. Iz nje elektroni odlaze u uređaj spojen u strujni krug. Anoda je označena kao minus (-) na bateriji, a kemijska reakcija između anode i elektrolita uzrokuje nakupljanje elektrona u anodi koji se žele premjestiti na katodu, ali ne mogu proći kroz elektrolit.

Katoda je dio baterije koji je označen kao plus (+). Pri spajanju u strujni krug, elektroni ulaze u katodu. Kemijska reakcija u katodi koristi elektrone koji su proizvedeni u anodi. Jedini način da elektroni dođu do katode je kroz strujni krug u kojem se baterija nalazi.

Elektrolit je tvar koja prenosi ione između kemijskih reakcija koje se događaju na anodi i katodi. Također služi za sprječavanje protoka elektrona između anode i katode kako bi usmjerio elektrone kroz vanjski strujni krug.

Baterije u sebi često imaju tvari koje sprječavaju anodu i katodu da se dodiruju jer bi njihov dodir značio kratki spoj u bateriji. Važno je da te tvari ne reagiraju sa anodom, katodom

i elektrolitom. Kućišta baterija mogu biti izrađena od bilo čega, a u slučaju AA alkalnih baterija, kućište je čelično i spojeno je s katodom. [6]

3.2.3. Način rada

Baterije stvaraju elektrone kemijskim reakcijama. Najmanje po jedna reakcija dogodi se u anodi i katodi. Reakcija na anodi stvara dodatne elektrone i taj proces se naziva oksidacija. S druge strane baterije na katodi događa se reakcija koja koristi te elektrone, a naziva se redukcija.

Oksidacija se događa između elektrolita i anode i stvara elektrone. Također, u nekim slučajevima kao što su litij-ionske baterije, proizvode ione. Između katode i elektrolita stvara se redukcija koja troši elektrolite. U litij-ionskim baterijama tijekom redukcije troše se pozitivno nabijeni ioni nastali u oksidaciji.

Kada kemikalije prestanu reagirati, baterija postaje prazna. Primarne baterije moraju se odložiti, dok se sekundarne ponovno mogu napuniti reverznom električnom strujom kroz bateriju. [6]

3.2.4. Terminologija

Nominalni napon baterije je napon koji je naveden na samoj bateriji, a mjeri se u voltima. U našem slučaju se elektromotori napajaju sa 4 AA baterije nominalnog napona 1.5V što bi u serijskom spoju značilo ukupno 6V snage. Raspberry Pi se napaja eksternom baterijom za punjenje na kojoj je nominalni napon 5V.

Kapacitet baterije mjeri količinu električnog naboja koji može isporučiti pod određenim naponom. Drugim riječima, određuje koliko dugo baterija može trajati. Baterija koja napaja Raspberry Pi ima kapacitet od 2200mAh dok baterije za pokretanje elektromotora imaju oko 2400mAh

Baterije mogu biti s jednom ćelijom, spojene paralelno, serijski ili oboje. U slučaju serijskog spoja, stvaramo veći napon, dok kapacitet ostaje nepromijenjen. Ako baterije spojimo paralelno, povećavamo kapacitet, a napon ostaje isti. Sa paralelnim spajanjem treba biti pažljiv jer nejednak napon može dovesti do kratkih spojeva baterija. Paralelni i serijski spojevi mogu se kombinirati na različite načine ovisno o tome što nam treba u zadanom trenutku. [6]

3.3. Elektromotori

Elektromotori su strojevi koji pretvaraju električnu u mehaničku energiju, dijele se u dvije kategorije: motori pokretani istosmjernom strujom (DC motori) te motori pokretani izmjeničnom strujom (AC motori) koji se još mogu podijeliti u sinkrone i asinkrone motore.

Sinkrone motore obilježava okretanje osovine motora jednakom brzinom kao i rotiranje magnetskog polja, dok kod asinkronih motora okretanje osovine sporije nego rotiranje magnetskog polja. [7]

Elektromotori su u osnovi obrnuti generatori, odnosno, struja kroz zavojnice pokreće mehanički objekt. Osnovni princip rada elektromotora je elektromagnetska indukcija. To znači da struja inducira magnetsko polje koje stvara interakciju s drugim magnetskim poljem što uzrokuje mehaničko kretanje.

Elektromotor kakav danas poznajemo otkriven je potpuno slučajno, 1873. godine kada je jedan DC generator slučajno spojen u drugi. Motore pokretane izmjeničnom strujom izumio je Nikola Tesla 1880ih godina. Obzirom da su zapravo obrnuti generatori, elektromotori se sastoje od rotora i statora te tri komponente: električne struje, magnetskog polja i nečega što se fizički okreće. [8]

3.3.1. Dijelovi motora

Elektromotori rade na princip elektromagnetizma. Struja koja prolazi kroz žicu stvara magnetsko polje, a ako je žica namotana oko šipke stvara se magnetsko polje oko osovine. Jedan kraj šipke tada će imati sjeverni, a drugi južni magnetski pol. Suprotni polovi se privlače, a magneti oko šipke vrte šipku privlačnim i odbojnim magnetnim silama.

Dva osnovna dijela elektromotora su rotor i stator. Stator je dio motora koji se ne pomiče nego sadržava magnet koji pomiče šipku. Rotor se sastoji od već spomenute šipke sa namotanom žicom. Kada struja teče kroz zavojnicu, magnetsko polje rotora se gura protiv polja koje stvara stator i osovina se okreće.

Komutator je dio motora koji se nalazi na jednom kraju zavojnice. To je obično metalni prsten podijeljen u dvije polovice. Njegov zadatak je da preokreće električnu struju svaki put kad se zavojnica omota oko pola okretaja. Taj proces je neophodan jer zapravo čini da se motor nastavi okretati. Da bi se rotor i dalje vrtio, komutator okreće polaritet magneta te ponovno dolazi do odbijanja i samim time okretanju motora.

Na kraju motora suprotnom od mjesta na kojem rotor izlazi iz kućišta, nalaze se četkice i stezaljke. Četkice šalju električnu struju na komutator dok su stezaljke mjesta na kojima bateriju pričvršćujemo na motor odnosno napajanje samog motora. [9]

3.3.2. Najčešći tipovi motora

Već smo spomenuli kako postoje motori sa istosmjernom i sa izmjeničnom strujom. Svaki od njih ima svoje potkategorije te svaka izvedba ima svoje prednosti i mane. Izbor tipa elektromotora ovisi o tome za što će se koristiti.

Među motorima na izmjeničnu struju nalaze se sinkroni motori. U toj vrsti motora rotacija rotora sinkronizirana je sa frekvencijom opskrbne struje, dok brzina ostaje konstantna pri različitim opterećenjima. Takav način rada idealan je za pogon opreme konstantnom brzinom i koristi se u uređajima za precizno pozicioniranje.

Drugi tip motora na izmjeničnu struju naziva se indukcijski motor. Indukcijski motor koristi elektromagnetsku indukciju iz magnetskog polja statorskog namota za proizvodnju struje u rotoru. To je najčešći tip motora na izmjeničnu struju. Na kućanskim aparatima koriste se uglavnom jednofazni, dok se trofazni indukcijski motori koriste za uređaje poput kompresora, pumpi i slično.

Motori koji koriste istosmjernu struju su drugi oblik motora. Razlikuju se po tome imaju li četkice ili ne. Motori s četkicama obično se koriste kada je cijena važan faktor i gdje je upravljački sustav motora relativno jednostavan. Prvi pod nazivom Series Wound, gdje se brzina kontrolira naponom, a koristi se za sve vrste dizala jer ima jak početni moment. Shunt Wound je drugi tip motora s četkicama koji ima konstantnu brzinu, a koristi se za usisivače, brusilice i slično. Treći oblik je Compund Wound koji je kumulativ Series Wound-a i Shunt Wound-a. Obično se koristi za pogon kompresora, kružnih pila, strojeva za rezanje i slično.

Motori bez četkica rješavaju problem dugotrajnosti i mehanički su mnogo jednostavniji. Upravljač motora koristi senzore za otkrivanje položaja rotora i pomoću njih regulator može precizno upravljati motorom putem struje u zavojnicama rotora. Velika prednost je i visoka učinkovitost tog motora, a nedostatak je cijena i složeniji kontroleri. Ovakve vrste motora koriste se u regulaciji brzine i položaja. [10]

4. Java programski jezik

Java je objektno orijentirani programski jezik kojim se proizvodi softver za više platformi. Kad programer napiše Java aplikaciju, kompajlirani kôd (poznat kao bytecode) radi na većini operativnih sustava (OS), uključujući Windows, Linux i Mac OS. Veliki dio Java sintakse potječe iz programskih jezika C i C ++.

Javu je sredinom 1990-ih razvio James A. Gosling, bivši informatičar iz Sun Microsystems, zajedno s Mikeom Sheridanom i Patrickom Naughtonom. [11]

4.1. Gdje se Java koristi?

Javina glavna karakteristika je neovisnost platforme. Za pokretanje Java programa treba samo Java Runtime Environment (JRE), bez obzira je li instalirana na stolno računalo sa sustavom Windows, Linux ili Unix, Macintosh računalo, pametni telefon i slično. U nastavku je nabrojano nekoliko uobičajenih mjesta na kojima se koristi.

1. Android aplikacije: Većina Android aplikacija koristi Java API ili su napisane na Javi, toliko da se Java često smatra službenim programskim jezikom za razvoj mobilnih aplikacija.

2. Desktop GUI aplikacije: Mnogo desktop aplikacija razvijeno je u Javi. Swing, Abstract Windowing Toolkit (AWT) i JavaFX glavni su alati koji se koriste za jednostavan razvoj grafičkog sučelja.

3. Web aplikacije: Java se često koristi za razvoj širokog spektra interaktivnih web stranica i web aplikacija.

4. Financijske i maloprodajne usluge: Java se koristi za pisanje aplikacija za upravljanje transakcijama i naplatu, kao i aplikacija na poslužitelju.

5. Znanost i istraživanje: Java je omiljeni jezik znanstvene zajednice za široki spektar matematičkih izračuna i drugih znanstvenih operacija. Može se nositi s ogromnim skupovima podataka i tehnologijama velikih podataka jer se koristi za MATLAB i Hadoop MapReduce sučelje. [11]

Iako je Python najčešće prvi izbor za RaspberryPi programiranje, za ovaj projekt je izabrana JAVA radi autorovog proširivanja znanja na trenutnom radnom mjestu.

4.2. Povijest Java programskog jezika

Javu je kao projekt nazvan "Oak" započeo James Gosling u lipnju 1991. Goslingovi ciljevi bili su implementirati virtualni stroj i jezik koji će biti sličan C jeziku, ali s većom ujednačenošću i jednostavnošću. Prva javna implementacija bila je Java 1.0 1995. godine, a glavni postulat bio je "Write once, run anywhere" što bi značilo da se program napisan u Java jeziku može izvršavati na bilo kojem sustavu. Java kao programski jezik bio je prilično siguran i njegova se sigurnost mogla konfigurirati, što je omogućilo ograničavanje pristupa mreži i datotekama.

Sun se 1997. obratio tijelu za norme ISO / IEC JTC1, a kasnije i Ecma International-u kako bi formalizirao Javu, ali je ubrzo odustao od procesa. Java ostaje vlasnički de facto standard koji se kontrolira kroz Java zajednicu. Sun većinu svojih Java implementacija čini dostupnim bez naknade, a prihod generiraju specijalizirani proizvodi poput Java Enterprise System. Sun razlikuje svoj Komplet za razvoj softvera (SDK) i Java pokretačko okruženje, odnosno Java Runtime Environment (JRE) koji je podskup SDK-a, a primarna razlika je u tome što u JRE kompajler nije prisutan. [12]

4.3. Osnovne značajke Jave

Razvoj jezika Java imao je pet primarnih ciljeva:

1. Objektno orijentirani pristup.
2. Omogućiti da više operativnih sustava pokreće isti program.
3. Imati ugrađenu podršku za računalnu mrežu.
4. Biti konfiguriran za sigurno izvršavanje koda iz udaljenih izvora.
5. Jednostavan za upotrebu. [12]

4.3.1. Objektno orijentiran

Prvi element, objektno orijentiran (OO), odnosi se na proces programiranja i dizajniranja. Iako je moguće nekoliko značenja OO, jedan od kritičnih principa razlikovanja je dizajn programa koji kombinira različite tipove podataka, a kojima manipulira s njihovim odgovarajućim operacijama. Podaci i kod zatim se kombiniraju u cjeline poznate kao objekti.

Entitet se može smatrati dijelom (kodom) i stanjem samostalnog ponašanja (podaci). Ideja je odvojiti predmete koji se mijenjaju od stvari koje ostaju iste; promjena u bilo kojoj

strukturi podataka također znači da se kôd koji radi na tim podacima ili obrnuto modificira u skladu s tim. Ova podjela na kompatibilne objekte nudi pouzdaniju osnovu za dizajn softverskog sustava. Cilj je promovirati upravljanje velikim softverskim projektima, povećati kvalitetu i smanjiti broj neuspjelih projekata.

Sljedeći je primarni cilj OO-a stvoriti uobičajenije artefakte kako bi softver između projekata mogao biti višekratno korišten. Na primjer, generički objekt „kupac“ trebao bi imati u osnovi isti osnovni skup ponašanja, značajno kada se ti projekti preklapaju na određenoj temeljnoj razini, kao što se to često događa u većim organizacijama.

Nadamo se da se u tom smislu softverski objekti mogu više tretirati kao plug-in komponente tako da softverska industrija može primarno graditi projekte na temelju trenutnih, dobro testiranih dijelova, čime se značajno smanjuje vrijeme razvoja. Ponovna upotreba softvera polučila je mješovite funkcionalne ishode s dvije primarne poteškoće: koncepcija i komunikacijski pristup stvarno generičkih objekata slabo je razumljiv. [12]

4.3.2. Neovisnost o platformi

Druga značajka, neovisnost o platformi, znači da Java napisani programi moraju jednako raditi na različitim hardverima. treba biti sposoban za pisanje jednom i izvršavanje programa bilo gdje.

Kôd se zatim izvodi na virtualnom stroju koji interpretira i izvršava generički Java bajt kôd u izvornom kodu na hardveru sustava koji ga pokreće. Također, dostupne su standardizirane knjižnice koje omogućuju jedinstveni pristup značajkama računala domaćina, poput grafike umrežavanja itd.

Sun-ova Java licenca zahtijeva "kompatibilnost" svih implementacija. To je dovelo do pravnog spora s Microsoftom. Nakon toga Microsoft više ne isporučuje Javu sa sustavom Windows, a Internet Explorer više ne podržava Java applet-e bez dodataka treće strane.

Međutim, za te i druge verzije sustava Windows, Sun i drugi učinili su Java sustave dostupnima besplatno. [12]

4.3.3. Upravljanje memorijom

Jedan koncept Javinog automatiziranog modela upravljanja memorijom je uštedjeti programerima stres ručnog upravljanja memorijom. Na određenim jezicima, programer dodjeljuje memoriju za izgradnju bilo kojeg objekta pohranjenog na hrpi, a zatim ručno upravlja memorijom kako bi ih uklonio. Do curenja memorije može doći kada programer zaboravi premjestiti memoriju i ne upiše kod odmah: troši potencijalno veliku količinu memorije. Nadalje, program može postati nestabilan i propasti kada se memorijska regija proširi dva puta.

Automatsko prikupljanje smeća izbjegava ovaj mogući problem na Javi. Kad se objekti kreiraju, programer odlučuje, a Java je odgovorna za upravljanje životnim ciklusom objekta. Držeći referencu, program ili drugi objekti mogu spominjati objekt (koji je s gledišta niske razine njegova adresa na hrpi). Java sakupljač smeća automatski briše nedostupni objekt ako ne ostane referenca na objekt, oslobađajući memoriju i izbjegavajući curenje memorije. [12]

5. Razvojni okviri i alati

5.1. Spring Boot

Spring Boot je open source mikro okvir koji održava tvrtka pod nazivom Pivotal. Pruža Java programerima platformu za početak rada s Spring aplikacijom koja se može automatski konfigurirati. Uz to, programeri mogu brzo započeti bez gubljenja vremena na pripremu i konfiguriranje svoje Spring aplikacije.

Spring Boot je izgrađen na temelju Spring okvira i dolazi s mnogim ovisnostima koje se mogu priključiti na Spring aplikaciju. Neki primjeri su Spring Kafka, Spring LDAP, Spring Web Services i Spring Security. [13]

5.1.1. Prednosti Springa i Spring Boota

Spring okvir se fokusira na pružanje fleksibilnosti kroz svoju značajku „dependency injection“. Pomaže brzo ubaciti potrebne „dependency“, ali i razviti aplikaciju na „loosely coupled“ način. Neke druge pogodnosti uključuju:

- Spring je programski lagan okvir.
- Ima podršku za XML i konfiguraciju preko anotacija.
- Pruža apstrakciju na ORM softveru za razvoj logike postojanosti ORM-a.
- Kompatibilan s mnogim Middleware uslugama.
- Podržava JDBC okvir, koji poboljšava produktivnost i smanjuje pogreške

Spring Boot, s druge strane, je fokusiran na skraćivanje duljine koda i pruža jednostavan način za pokretanje Spring aplikacije. [13]

5.1.2. Korištenje Spring Boot okvira u projektu

U ovom projektu, najbitnija stavka Spring Boot-a je jednostavnost kreiranja aplikacije u kojoj će se koristiti Spring da bi komponente mogle biti kreirane s anotacijama. Jedna od komponenti će biti kontroler koji pretvara signal s tipkovnice u naredbu. Druga komponenta biti će sam robot koji prima naredbu i pretvara ju u pokret.

5.1.3. Način korištenja Spring Boot okvira

Komponenta se radi tako da se kreira klasa pod nazivom Robot i doda se anotacija @Component. Nakon što je to napravljeno, Spring dodaje klasu Robot među svoje „beanove“ i kasnije instancu te klase korisnik može koristiti tako da doda polje označeno s @Autowired u nekoj drugoj klasi anotiranoj s @Component.

5.2. Maven

Bez obzira koliko male ili velike, sve aplikacije moraju proći kroz specifičan niz procesa, kao što je generiranje i prevođenje izvornog koda. Programeri mogu ručno konfigurirati ove procese, ali to je dugotrajan posao.

Kako bi ti procesi postali automatizirani, u ovom projektu se korist Apache Maven, koji automatizira cijeli proces i olakšava svakodnevni rad Java programera.

Maven je popularan alat otvorenog koda koji je razvio Apache Group za izgradnju, objavljivanje i implementaciju nekoliko projekata odjednom radi boljeg upravljanja projektima. Alat omogućuje programerima da izgrade i dokumentiraju okvir životnog ciklusa.

Napisan je u Javi i koristi se za izgradnju projekata napisanih na C#, Scala, Ruby, itd. Na temelju Project Object Modela (POM), ovaj alat je olakšao život Java programerima dok pišu izvješća, provjeravaju izgradnju i testiraju automatizaciju postavke.

Usredotočuje se na pojednostavljenje i standardizaciju procesa izgradnje, vodeći računa o sljedećem:

- Gradnji
- Dokumentaciji
- Ovisnostima (Dependency)
- Izvještajima
- SCM-ovima
- Distribucijom
- Izdanjima
- E-mail lista [14]

5.2.1. Korištenje Maven-a u ovom projektu

U ovom slučaju Maven se koristi kako bi lakše dodali ovisnosti (dependency), od kojih je jedna Pi4J koja služi za kontroliranje samog Raspberry Pi uređaja. Također, Maven se koristi u ovom projektu da kreira .jar datoteku koja se pokreće na samom Raspberry Pi računalu.

5.2.2. Način korištenja Maven-a

Prvo, u već generirani pom.xml u projektu dodaje se ovisnost (dependency) za Pi4J na ovaj način:

```
<dependency>
    <groupId>com.pi4j</groupId>
    <artifactId>pi4j-core</artifactId>
    <version>1.4</version>
</dependency>
```

Isto tako dodaje se i Spring Boot starter koji nam je potreban kako bismo mogli koristiti Spring Boot okvir:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
</dependency>
```

Nakon što je implementirano pokretanje robota, pokreće se naredba mvn clean install koja od projekta kreira .jar file koji se kasnije prebaci na Raspberry Pi uređaj koji taj kod pomoću java kompajlera prevodi i pretvara u aplikaciju kojom se može pokretati robot. Za proces pretvaranja u .jar file koji se može koristiti na RaspberryPi računalu koristimo plugin:

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

5.3. Korištene ovisnosti (dependency)

5.3.1. Pi4J

Pi4J je projekt koji nudi objektno orijentirane biblioteke za programski jezik Java kako bi se mogao iskoristiti puni potencijal Raspberry Pi računala. Projekt omogućava da se koristi native integracija s Raspberry Pi računalom pomoću Java koda kao što je recimo GPIO pin. [15]

6. Raspberry Pi

6.1. Što je Raspberry Pi?

Raspberry Pi je jeftino računalo veličine kreditne kartice koje se spaja na računalni monitor ili TV i koristi standardnu tipkovnicu i miša. To je sposoban mali uređaj koji ljudima svih dobnih skupina omogućuje istraživanje računalstva i učenje programiranja na jezicima kao što su Scratch i Python. Može učiniti sve što očekujete od stolnog računala, od pregledavanja interneta i reprodukcije videozapisa visoke razlučivosti, do izrade proračunskih tablica, obrade teksta i igranja igrica.

Štoviše, Raspberry Pi ima sposobnost interakcije s vanjskim svijetom i korišten je u širokom spektru projekata digitalnih proizvođača. „Želimo vidjeti kako djeca diljem svijeta koriste Raspberry Pi kako bi naučila programirati i razumjeti kako računala rade.“ [17]

Drugim riječima, Raspberry Pi nije ništa drugo nego obično računalo sa dodatnim GPIO konektorima kako bi moglo kontrolirati druge uređaje, elektromotore, robote i slično.

6.2. Povijest Raspberry Pi računala

Raspberry Pi uređaje razvila je dobrotvorna organizacija sa sjedištem u Velikoj Britaniji koja ima za cilj pružiti digitalno računarstvo ljudima u svim dijelovima svijeta. Većina škola i fakulteta preferira Raspberry Pi za opću namjenu. Međutim, Raspberry Pi ranije nije bio zamišljen kao dobrotvorni program. Mali tim računalnog laboratorija na Sveučilištu Cambridge otkrio je opadajući interes za računala zbog sve većih troškova i teškog održavanja tipičnih računalnih sustava. Odlučili su pronaći rješenje za ovaj problem i tako je 2012. godine rođen Raspberry Pi.

Kao što je gore spomenuto, Raspberry Pi je razvijen kao obrazovno računalo, ali njegova popularnost nije narasla zbog same cijene jer su si ljudi u to vrijeme već mogli priuštiti standardna računala. Raspberry je bio zapažen zbog svog koncepta jednopločnog računala (SBC – Single Board Computer).

Glavni članovi tima koji stoje iza osnivača Raspberry Pi-a su Eben Upton, Rob Mullins, Jack Lang i Alan Mycroft.

Prvo komercijalno izdanje Raspberry Pi objavljeno je u veljači 2012. Prvi SBC bio je Raspberry Pi Model B. Ovaj model je na tržište izašao 15. veljače 2012. po cijeni od 35 dolara. Bilo je to računalo s jednojezgrenim procesorom, 512MB radne memorije te s dva 2.0 USB priključka. Nije imao SATA portove nego HDMI priključak. Analogni zvuk se mogao

reproducirati putem GPIO pinova ili USB zvučne kartice. Ovo računalo brzo je steklo popularnost jer je bilo prilično jeftinija zamjena za opća računala. [18]

6.3. Korišteni model u ovom projektu

U ovom projektu korišten je Raspberry Pi Model 4 B. U njemu se nalazi:

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz procesor
- 2GB LPDDR4-3200 SDRAM radna memorija
- 2.4 GHz i 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 USB 3.0 porta; 2 USB 2.0 porta.
- Raspberry Pi standardno 40 pin GPIO sučelje
- 2 × micro-HDMI portovi (s podrškom za 4k na 60hz)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-polni port za stereo zvuk i video
- Micro-SD card slot za operativni sustav i pohranu
- USB-C konektor za napajanje [19]

Također za potrebe kreiranja vozila korišten je i CamJam EduKit 3 – Robotics koji sadržava:

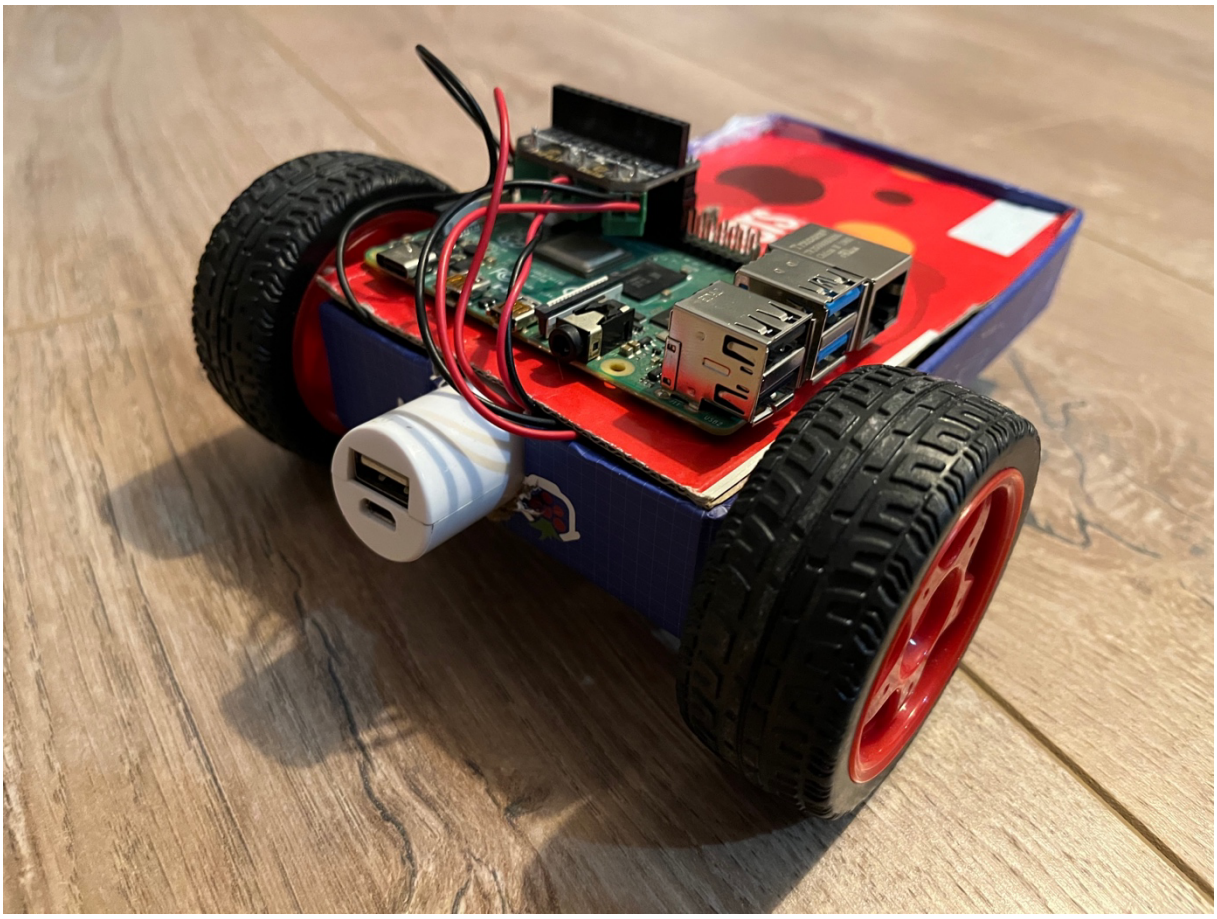
- Kontroler za elektromotore koji se direktno stavlja na GPIO pinove
- Dva DC motora
- Dva kotača
- Kuglica u nosaču koja služi kao treći kotač
- Držač za 4 AA baterije
- Kablovi za povezivanje [20]

CamJam EduKit sadržava i još neke dijelove koji nisu korišteni za potrebe ovog projekta.

7. Opis gotovog vozila

7.1. Fizički dijelovi

Ovaj robot se pokreće bežičnom tipkovnicom koja je povezana na Raspberry Pi Model 4 B. Pritisak na tipku pokreće kasnije opisani program koji je aktivan na Raspberry Pi računalu. Program daje signal u kontroler motora iz CamJam Edukit-a. Nakon što primi signal, kontroler dohvaća električnu energiju iz 4 AA baterije koje se nalaze u kućištu te pokreće elektromotor koji vrti kotače. Kotači su pričvršćeni na kutiju od CamJam Edukit-a koja služi kao tijelo robota. U kućištu se također nalazi i prijenosni punjač koji napaja Raspberry Pi.



Slika 1. Krajnji proizvod

7.2. Programska podrška

U ovom poglavlju analizirati ćemo kompletan kod koji se koristi kako bi se na pritisak tipke na tipkovnici, naš robot pomaknuo.

7.2.1. RaspberryPi Application

Kao i svaka Java aplikacija, ova aplikacija pokreće se main metodom. Obzirom da koristimo Spring Boot kao okvir, početna klasa izgleda ovako:

```
@SpringBootApplication
public class RaspberryPiApplication {

    public static void main(String[] args) {
        SpringApplication.run(RaspberryPiApplication.class, args);
    }
}
```

Ovdje se vidi da je klasa anotirana sa @SpringBootApplication anotacijom kojom se označuje main klasa u Spring aplikacijama. Ova klasa sadržava jednu metodu koja se naziva main, a ima standardni potpis za Java glavnu metodu a to je: public static void main (String [] args), a u njoj se poziva metoda statičke klase SpringApplication run(). U nju se kao parametre šalje glavna klasa i argumenti koji se prosljeđuju prilikom pokretanja aplikacije.

7.2.2. KeyboardConfig i RaspberryConfig

Kako bi se moglo jednostavno odrediti koja tipka je za koji pokret robota i kasnije to mijenjati bez da se mijenja kod, u application.properties se nalaze sljedeće postavke:

```
keyboard.forward=up
keyboard.backward=down
keyboard.left=left
keyboard.right=right
robot.rightBackward=10
robot.rightForward=11
robot.leftBackward=12
robot.leftForward=13
```

Ovom konfiguracijom je određeno da će za pokretanje vozila prema naprijed služiti tipka Up odnosno strjelica prema gore, prema nazad strjelica prema dolje i tako dalje. Kako bi se ta konfiguracija mogla koristiti, potrebno je napraviti klasu KeyboardConfig koja te parametre mapira u Spring Boot aplikaciju.

```
@Configuration
@ConfigurationProperties(prefix = "keyboard")
public class KeyboardConfig {
    private String forward;
    private String backward;
    private String left;
    private String right;

// getteri i setteri

private String capitalizeFirst(final String str) {
    return str.substring(0, 1).toUpperCase() + str.substring(1);
}
```

Kako bi se mogla koristiti application.properties datoteka, potrebno je klasu anotirati sa @Configuration i @ConfigurationProperties. Prefix keyboard označava prefix koji postoji i u application.properties datoteci. Imena polja moraju odgovarati drugom dijelu postavki, onom iza točke, iz application.properties datoteke. Ovdje se nalazi i metoda capitalizeFirst koja se poziva na svaki setter kako bismo izbjegli greške u konfiguracijskoj datoteci. Naime, kod koji prima ulaz s tipkovnice svaku tipku na tipkovnici gleda kao veliko početno slovo tako da se u application.properties datoteci može pisati konfiguracija malim slovima. Nakon što se aplikacija pokrene, vrijednosti ovih polja se postave i mogu se koristiti dalje u aplikaciji.

```
@Configuration
@ConfigurationProperties(prefix = "robot")
public class RaspberryConfig extends RaspiPin {
    private int rightForward;
    private int rightBackward;
    private int leftForward;
    private int leftBackward;

    public int getRightForward() {
        return rightForward;
    }

    public void setRightForward(int rightForward) {
        this.rightForward = rightForward;
    }
```

Na ovaj način se dohvaćaju postavke za robota, odnosno na kojem GPIO pinu se nalazi koji pokret elektromotora.

7.2.3. RobotController

Druga klasa je RobotController koja pretvara pritiske tipki na tipkovnici u naredbe robotu.

```
@Component
public class RaspberryRobotController implements KeyListener {

    private final Map<String, Runnable> map;
    private final Robotic robotic;

    @Autowired
    public RaspberryRobotController(final KeyboardConfig config, final
Robotic robotic) throws InterruptedException {
        Frame f = new Frame();
        f.setVisible(true);
        f.addKeyListener(this);
        map = new HashMap<>();
        map.put(config.getForward(), robotic::forward);
        map.put(config.getBackward(), robotic::backward);
        map.put(config.getLeft(), robotic::left);
        map.put(config.getRight(), robotic::right);
        this.robotic = robotic;
        robotic.forward();
        Thread.sleep(500);
        robotic.neutralize();
    }
```

```
    @Override
    public void keyPressed(KeyEvent e) {
        map.getOrDefault(KeyEvent.getKeyText(e.getKeyCode()), () -> {
        }).run();
    }

    @Override
    public void keyReleased(KeyEvent e) {
        robotic.neutralize();
    }

    @Override
    public void keyTyped(KeyEvent e) {
    }
}
```

RobotController klasa, koja implementira KeyListener sučelje, označena je sa @Component kako bi postala dio Spring konteksta. Kada je klasa dio Spring konteksta, njena implementacija može se koristiti u ostalim klasama, isto tako, otvara mogućnost korištenja drugih implementacija klasa iz konteksta.

Kako bi se mogli pratiti pritisci tipki na tipkovnici, implementirano je KeyListener sučelje koje, između ostalih, u sebi sadrži metode keyPressed() i keyReleased().

Na početku su deklarirana dva polja: Map<String, Runnable> map i Robotic robotic, kojima se u konstruktoru klase kreira vrijednost.

Jedini konstruktor ove klase anotiran je sa @Autowired što znači da se svi njegovi ulazni parametri dohvaćaju iz, već ranije spomenutog, Spring konteksta. KeyboardConfig i Robotic su klase koje su anotirane sa @Component tako da ih je zbog toga moguće dohvatiti iz Spring konteksta. Kako bi tipkovnica radila sa Java aplikacijom, potrebno je napraviti Frame objekt pomoću njegovog konstruktora bez parametara. Zatim je u njega potrebno staviti polje visible u true i dodati implementaciju KeyListener sučelja. Nakon toga, polju map je dodana vrijednost pomoću poziva konstruktora new HashMap<>(), te su u tu mapu dodani parovi koji povezuju tipku na tipkovnici sa stvarnim pokretom robota. Pri kraju samog konstruktora, dodana je vrijednost Robotic sučelja polju robotic dok je na samom kraju pozvana metoda koja pokreće sam robot kako bi smo znali u kojem je trenutku spreman za korištenje.

Sljedeće dvije metode su dio KeyListener sučelja a zovu se keyPressed() i keyReleased(). Te dvije metode se pokreću prilikom svakog pritiska ili puštanja tipke. Obje su označene sa @Override kako bi bilo naglašeno da su dio implementiranog sučelja,

Metoda keyPressed() se pokreće prilikom pritiskanja tipke. Kada se tipka pritisne, dio koda KeyEvent.*getKeyText*(e.getKeyCode()) vraća tekstualnu vrijednost tipke koja je pritisnuta. Ta vrijednost se nakon toga traži kao ključ u mapi koju smo povezali sa metodama za pokretanje robota. Kako bi metoda bila pozvana, potrebno je nad vraćenim objektom pozvati metodu run().

Metoda keyReleased() služi kako bi spriječila nekontrolirano kretanje robota. Prilikom svakog puštanja tipke, robotu će biti poslana naredba da neutralizira kretnju.

Metoda keyTyped() nema konkretnu implementaciju jer nije bila potrebna, ali ju je bilo potrebno fiktivno implementirati zbog toga što se nalazi u sučelju,

7.2.4. Robotic, DummyRobot, RaspberryPiRobot

Obzirom da se na računalu koje nije Raspberry Pi ne može pokrenuti konkretna implementacija iz Pi4J biblioteke, kako bi implementacija i testiranje bili jednostavniji, bilo je potrebno kreirati dvije odvojene implementacije za pokretanje robota. RaspberryPiRobot za stvarno kretanje robota te DummyRobot za ispisivanje kretanja u konzolu radi jednostavnijeg programiranja ostalih elemenata aplikacije. Kako bi mijenjanje konkretnog koda obzirom na uređaj na kojem se aplikacija pokreće bilo izbjegnuto, koriste se Spring profili. Prvo je kreirano Robotic sučelje:

```
public interface Robotic {

    void forward();

    void backward();

    void left();

    void right();

    void neutralize();

}
```

Ovo jednostavno sučelje govori koje metode robot mora implementirati. Implementacija DummyRobot klase izgleda ovako:

```
@Component
@Profile("default")
public class DummyRobot implements Robotic {

    @Override
    public void forward() {
        System.out.println("Forward");
    }

    @Override
    public void backward() {
        System.out.println("Backward");
    }

    @Override
    public void left() {
        System.out.println("Left");
    }

    @Override
    public void right() {
        System.out.println("Right");
    }
```

```
    @Override
    public void neutralize() {
        System.out.println("Stop");
    }

}
```

Klasa je anotirana s anotacijom @Component kako bi bila dio springovog konteksta, a anotacija @Profile(„default“) govori da je to implementacija default profila, odnosno profila koji se koristi ako se ne navede niti jedan profil. Ova klasa služi kako bi olakšala implementaciju drugih dijelova koda. Kao što se može vidjeti, klasa implementira Robotic sučelje, a sve njegove metode implementira tako što u konzolu ispiše koji pokret robot treba raditi.

Prava implementacija kretnje robota krije se u klasi RaspberryPiRobot koja također implementira Robotic sučelje. @Profile anotacija ima vrijednost prod što znači da je potrebno aplikaciju pokrenuti sa profilom prod kako bi koristila ovu implementaciju Robotic sučelja.

```
@Component
@Profile("prod")
public class RaspberryPiRobot implements Robotic {
    private final GpioController gpio;
    private final GpioPinDigitalOutput rightBackward;
    private final GpioPinDigitalOutput rightForward;
    private final GpioPinDigitalOutput leftBackward;
    private final GpioPinDigitalOutput leftForward;

    @Autowired
    public RaspberryPiRobot(final RaspberryConfig raspberryConfig) {
        // create gpio controller
        this.gpio = GpioFactory.getInstance();

        rightForward = createGPIO(raspberryConfig.getRightForward(),
"rightForward");
        rightBackward = createGPIO(raspberryConfig.getRightBackward(),
"rightBackward");
        leftForward = createGPIO(raspberryConfig.getLeftForward(),
"leftForward");
        leftBackward = createGPIO(raspberryConfig.getLeftBackward(),
"leftBackward");
    }

    @Override
    public void forward() {
        System.out.println("Moving raspberry forward");
        rightForward.high();
        leftForward.high();
    }

    @Override
    public void backward() {
        System.out.println("Moving raspberry backward");
        rightBackward.high();
        leftBackward.high();
    }

    @Override
    public void left() {
        System.out.println("Moving raspberry left");
```

```
        leftBackward.high();
        rightForward.high();
    }

    @Override
    public void right() {
        System.out.println("Moving raspberry right");
        rightBackward.high();
        leftForward.high();
    }

    @Override
    public void neutralize() {
        System.out.println("Neutralizing raspberry");
        rightForward.low();
        rightBackward.low();
        leftBackward.low();
        leftForward.low();
    }

    private GpioPinDigitalOutput createGPIO(Integer pinNumber, String
position) {
        GpioPinDigitalOutput output =
gpio.provisionDigitalOutputPin(getPinByName("GPIO " + pinNumber), position,
PinState.LOW);
        return output;
    }

}
```

Na početku klase se nalaze polja tipa GpioPinDigitalOutput koja služe kako bi se mogli aktivirati pojedini GPIO pinovi na Raspberry Pi računalu.

U konstruktoru koji prima RaspberryConfig konfiguracijsku klasu na početku je kreiran GpioController objekt s kojim se kreiraju i GpioPinDigitalOutput objekti koji su referencirani na polja deklarirana na početku klase. GpioPinDigitalOutput objekti se kreiraju, između ostalog, iz objekta tipa RaspberryConfig.

Kako i ova klasa implementira Robotic sučelje, mora implementirati metode koje se nalaze u sučelju, kao što su forward, backward, neturalize i druge. Da bi se robot pokrenuo naprijed, potrebno je i lijevi i desni kotač pokrenuti naprijed. To se može postići tako da se nad objektima rightForward i leftForward pokrene metoda high() koja uključuje elektromotor mapiran na konkretan GPIO pin. Logično, robot se pokreće u natrag pozivanjem metode high() nad rightBackward i leftBakcward objektima. Okretanje robota u desnu stranu odvija se tako da se desni kotač okreće u natrag, a lijevi u naprijed. Samim time, treba pozvati metode high() nad objektima rightBackward i leftForward. Okretanje robota u lijevu stranu, odvija se metodom high() na preostalim objektima. Već ranije spominjana metoda neutralize() služi kako bi se svi GPIO pinovi neutralizirali. To se može napraviti pokretanjem metodom low() nad svim GpioPinDigitalOutput objektima. Metoda neutralize pokreće se kada se bilo koja tipka pusti.

7.3. Priprema Raspberry Pi računala

7.3.1. Instalacija Raspberry Pi OS sustava

Instalacija sustava Raspbian vrlo je jednostavan. Instalacijom aplikacije Raspberry Pi Imager vrlo jednostavno se odabere sustav koji se instalira te SD kartica na koju želimo instalirati sustav. Kartica sa sustavom se kasnije prebaci u sam Raspberry koji kasnije taj sustav zna pokrenuti.

7.3.2. Instalacija potrebnih alata

Kako bi Raspberry Pi mogao i znao izvršiti napisani kod, potrebno ga je pripremiti u nekoliko koraka. Za početak bilo je potrebno instalirati Java Development Kit kako bi Raspberry mogao izvršavati Java kod. To se može učiniti pokretanjem dvije naredbe:

```
sudo apt update
sudo apt install default-jdk
```

Obzirom da se nakon toga aplikacija nije mogla pokrenuti, bilo je potrebno dodati i Pi4J te wiringpi što se može učiniti sljedećim naredbama:

```
curl -sSL https://pi4j.com/install | sudo bash
sudo pi4j —wiringpi
```

Važno je napomenuti da sve ove naredbe zahtijevaju povezanost na Internet.

7.3.3. Pokretanje aplikacije

Aplikacija se pokreće pomoću naredbe:

```
java -jar -Dspring.profiles.active=prod Djava.awt.headless=false raspberry-pi-1.0.0.jar
```

Java -jar dio je glavni dio naredbe koji pokreće .jar datoteku. -Dspring.profiles.active i Djava.awt.headless su parametri koji su dodani kako bi aplikacija bila pokrenuta u prod profilu te kako bi se mogao pokrenuti dio koda koji prati ulaze s tipkovnice. Zadnji dio je ime kreirane .jar datoteke koju je izgenerirao Maven.

Nakon što je aplikacija pokrenuta i testirana, bilo je potrebno pripremiti i automatsko pokretanje aplikacije pri uključivanju računala kako bi se omogućilo jednostavno korištenje bez priključivanja na periferne uređaje:

Prvo je otvorena .bashrc datoteka pomoću naredbe: sudo nano .bashrc te je na kraju te datoteke bilo potrebno dodati naredbu za pokretanje aplikacije. Nakon toga, trebalo je pomoću naredbe: sudo nano /etc/xdg/lxsession/LXDE-pi/autostart otvoriti autostart datoteku i u nju dodati @lexterminal. Time je osigurano da će se aplikacija pokrenuti kada je pokrenuto Raspberry Pi računalo.

8. Zaključak

Robotika je u posljednje vrijeme postala vrlo dostupna upravo zbog računala poput Raspberry Pi. Iako je puno jednostavnije koristiti Python za pokretanje ovakvog robota, a i dokumentacija koja se nalazi na internetu uglavnom je vezana za Python, moguće je napraviti ovako jednostavan robot i u Java programskom jeziku. Spring Boot kao razvojni okvir je uvelike pomogao u jednostavnosti izrade implementacije ovog robota. Također, pomoću CamJam EduKit-a je izbjegnuto puno komplikacija s elektromotorima, kontrolerima i sličnim tako da je primarni fokus mogao biti na programiranju što i jest cilj ovog projekta.

Prilikom izrade ovog završnog rada korišteni su programi: IntelliJ Idea i Microsoft Word. Većinu metoda korištenih u praktičnom dijelu autor je naučio na trenutnom radnom mjestu te je isto znanje primijenio na ovaj praktični projekt. Također, sav kod nalazi se na GitHub repozitoriju https://github.com/FranKomlinovic/raspberry-pi

Pretpostavka da takav projekt neće biti posebno programski kompliciran je bila točna. Biblioteka Pi4J bila je od velike pomoći kako bi se izbjeglo kodiranje i korištenje native metoda. Pristup bez tih biblioteka bio bi vrlo kompliciran i pogrešan.

Najveća prepreka ovog projekta bilo je pokretanje aplikacije na samom Raspberry Pi računalu. Slaba dokumentacija i manjak informacija je uvelike otežao cijeli proces. Dobar primjer toga je to što ranije zamišljeno korištenje JNativeHook biblioteke nije bilo moguće zbog nepostojanja određenih native metoda na samom Raspberry-u.

Usprkos tome, svijet je danas došao do visoke razine u kojoj su ovakvi projekti relativno jednostavni za napraviti, a većina potrebnih komponenti već postoji, potrebno ih je samo spojiti i napraviti nešto novo. Također, razni alati i razvojni okviri napravili su od Jave relativno logičan i jednostavan jezik za bilo koju svrhu pa tako i za robotiku.

Na temelju svega što je već napravljeno i što je lako i jeftino dostupno, smatram da robotika i programiranje idu u dobrom smjeru i ostaje samo vidjeti na koji način će se ta područja zanimanja razviti i kakva ih budućnost čeka.

Popis literature

[1] J. Adam Morecki, "Basic of Robotics: Theory and Components of Manipulators and Robots".

[2] C. G. G. R. A. F. D. J. T. N. G. Hockstein, »A history of robots: from science fiction to surgical robotics«.

[3] L. S. A. Gasparetto, »A Brief History of Industrial Robotics in the 20th Century«.

[4] D. Cook, Robot Building for Beginners, Third Edition.

[5] C. Taylor, »Sparkfun,« [Mrežno]. Available: https://learn.sparkfun.com/tutorials/voltage-current-resistance-and-ohms-law.

[6] S. Hymel, »sparkfun,« [Mrežno]. Available: https://learn.sparkfun.com/tutorials/what-is-a-battery.

[7] T. Genberg, »Motor Fundamentals«.

[8] E. Moyer, »Basic on electric motors«.

[9] G. Beckett, »Sciencing,« 17 4 2018. [Mrežno]. Available: https://sciencing.com/parts-motor-5426656.html.

[10] »DesignSpark,« 5 5 2016. [Mrežno]. Available: https://www.rs-online.com/designspark/different-types-of-motors-and-their-use.

[11] J. Stoltzfus, »technopedia,« 27 11 2020. [Mrežno]. Available: https://www.techopedia.com/definition/3927/java.

[12] A. Sarangam, 30 1 2021. [Mrežno]. Available: https://www.jigsawacademy.com/blogs/java/history-of-java/.

[13] M. Mulders, 16 9 2019. [Mrežno]. Available: https://stackify.com/what-is-spring-boot/.

[14] I. Gaba, 28 10 2021. [Mrežno]. Available: https://www.simplilearn.com/tutorials/maven-tutorial/what-is-maven.

[15] Pi4J. [Mrežno]. Available: https://pi4j.com/.

[16] R. Pi, »Raspberry Pi,« [Mrežno]. Available: https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/.

[17] V. Dass. [Mrežno]. Available: https://linuxhint.com/raspberry-pi-history/.

[18] R. Pi. [Mrežno]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

[19] CamJam. [Mrežno]. Available: https://camjam.me/?page_id=1035.

[20] kwhat. [Mrežno]. Available: https://github.com/kwhat/jnativehook.

Popis slika