POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: AUTOMATYKA I ROBOTYKA
SPECJALNOŚĆ: SYSTEMY INFORMATYCZNE W AUTOMATYCE

PRACA DYPLOMOWA INŻYNIERSKA

Aplikacja do zdalnego zarządzania platformą mobilną

Application for remote control of mobile platform

AUTOR:
Piotr Jakubowski

PROWADZĄCY PRACĘ:
Dr inż. Krzysztof Halawa
Katedra Informatyki
Technicznej(W4/K9)

OCENA PRACY:

WROCŁAW 2018

SPIS TREŚCI:

1. Wstęp

Informatyka coraz bardziej ma wpływ na nasze życie codzienne oraz ma coraz szersze zastosowanie w dzisiejszym świecie. Stosowana jest w takich dziedzinach jak robotyka, medycyna, przemysł, lotnictwo oraz w użytku codziennym. Ostatnimi czasy na coraz większej popularności zyskują roboty ułatwiające nam wykonywanie codziennych działań. W dawnych czasach komputery stacjonarne zajmowały dużą przestrzeń, a ich moc obliczeniowa pozostawiała wiele do życzenia, a nawet małe operacje mogły trwać kilka godzin. Obecnie każdy z nas posiada w kieszeni małe urządzenie, które posiada wielokrotność mocy starych komputerów i możemy wykorzystywać je do różnych działań. Są to telefony komórkowe, bazujące na różnych systemach operacyjnych w zależności od ich producenta, lecz obecnie na rynku liczą się dwa duże systemy operacyjne:

- Android
- iOS

Platforma iOS została wykorzystana jedynie przez produkty tworzone przez firmę Apple w swoich telefonach komórkowych –IPhone. Jest to system zamknięty dostępny jedynie w tych konkretnych urządzeniach. Android zaś jest systemem wykorzystywanym przez większość największym firm produkujących telefony komórkowe jak Samsung, Huawei czy Motorola. Większość z tych urządzeń działają na Androidzie z wykorzystanymi przez konkretnego producenta własnych nakładek, lecz niektóre z nich (jak Motorola) działają na czystym Androidzie. Telefony pomagają nam w komunikacji pomiędzy użytkownikami, jak i komunikacji między człowiekiem a maszyną/urządzeniem.

Celem i tematyką obecnej pracy dyplomowej jest właśnie komunikacja między aplikacją napisaną na system Android, a platformą mobilną do sterowania manualnego oraz jazdy autonomicznej z wykorzystaną kamerą do przesyłania obrazu na żywo do aplikacji oraz czujnikami odległościowymi. W ramach części teoretycznej w rozdziale drugim została opisana sama budowa programu do tworzenia i kompilowania aplikacji mobilnych. W rozdziale trzecim zawarte zostały informacje odnośnie protokołu komunikacyjnego SSH wykorzystanego do komunikacji między urządzeniem a aplikacją mobilną. W rozdziale czwartym opisane zostały aspekty techniczne kamery użytej do przesyłania obrazu na żywo z platformy mobilnej do urządzenia mobilnego. Rozdział piąty zaś opisuje teoretycznie czym jest autonomia w robotyce oraz skupia się na jej ogólnym podziale ze względu na stopień autonomiczności. W rozdziale szóstym zostały udostępnione informacje odnośnie samej konfiguracji Raspberry Pi, czyli jej najważniejszych opcji wykorzystanych w projekcie, jak ustawianie statycznego IP, czy aktywowanie komunikacji SSH czy aktywowanie kamery. W rozdziale siódmym rozpisana została krok po kroku realizacja całego projektu wraz z jej najważniejszymi funkcjami, funkcjonalnościami oraz zaimplementowanymi rozwiązaniami. Wyszczególnione zostały najważniejsze funkcje oraz sama reprezentacja graficzna, pozwalająca użytkownikowi na wygodne korzystanie z aplikacji. Na końcu znajduje się podsumowanie całej pracy dyplomowej, wraz z wykorzystaną w pracy literaturą.

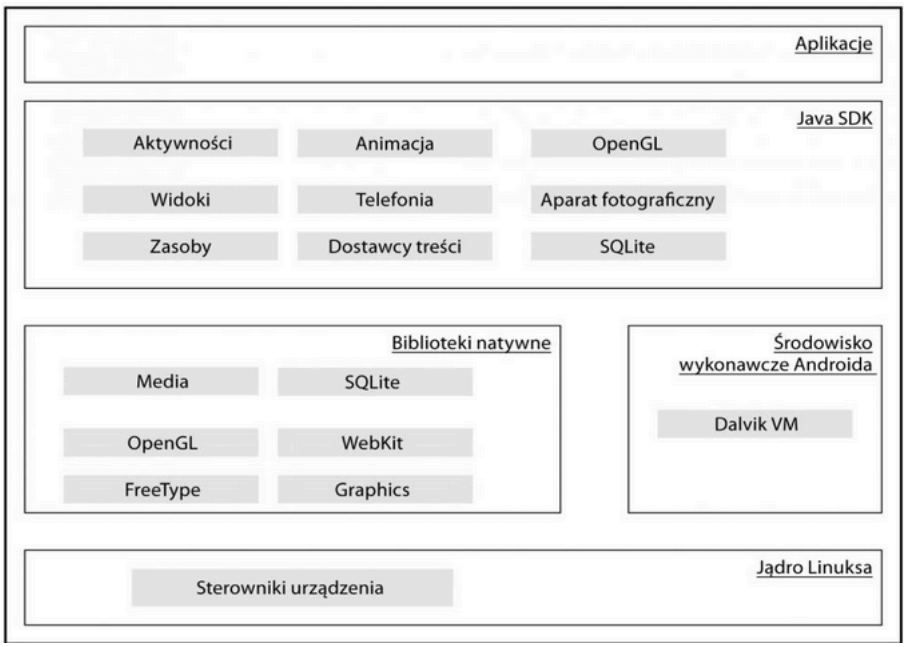
2. WGLĄD W WYKORZYSTANE ŚRODOWISKO ANDROID STUDIO.

2.1. Czym jest Android Studio.

Android Studio jest to środowisko służące do programowania aplikacji na Android, który został stworzony przez Google na bazie InteliiJ. Jest to najpopularniejsze środowisko na tę platformę pozwalające projektować, tworzyć oraz debugować własne programy dla aplikacji mobilnych.

Zestaw Android SDK (ang. Software Development Kit – zestaw do tworzenia oprogramowania) dostarcza możliwość obsługi znacznej części platformy Java Standard (Java SE), z wyjątkiem narzędzia Abstract Window Toolkit (AWT) oraz Swing – dwie główne biblioteki graficzne środowiska Java. Zamiast tych narzędzi w Android Studio został wykorzystany własny, obszerny i nowoczesny szkielet interfejsu użytkownika. Najbardziej popularnych językiem programowania aplikacji mobilnych na platformę Android jest do tej pory język Java, zatem niezbędne jest w nim środowisko JVM (ang. Java Virtual Machine – wirtualna maszyna Javy), w którym odbywa się proces interpretowania kodu bajtowego. Dzięki JVM otrzymujemy niezbędną optymalizację, pozwalającą osiągnąć wydajność porównywalną do wydajności aplikacji stworzonych w takich językach jak C oraz C++.

2.2. Opis wnętrza Androida.



Rysunek 2.1. Główne składniki platformy Android, źródło: S. Komatineni, D. MacLean, S. Hashimi „Android 3 – tworzenie aplikacji", s. 37.

Kernel Linuksa

We wnętrzu Androida znajduje się jądro Linuksa, które zapewnia standardową obsługę sterowników urządzenia, zarządzanie zasilaniem, możliwością dostępu do plików oraz głównymi zadaniami systemu operacyjnego. Sterowniki urządzenia obejmują ekran, aparat klawiaturę, Wi-Fi, pamięć flash, audio oraz komunikację IPC. Zarządza jednocześnie procesami oraz wątkami, gdzie proces jest uruchamiany dla każdej aplikacji. Dla procesu kod aplikacji może wykorzystać wiele wątków, przy czym to jądro rozdziela czas dla poszczególnych procesów i ich wątków dzięki planowaniu.

Android Runtime

Nad rdzeniem Linuksa, znajdują się biblioteki C oraz C++ (w ich skład wchodzą biblioteki: Open GL, WebKit, FreeType, SSL, SQLite oraz Media). W przypadku urządzeń z Androidem w wersji 5.0 lub wyższym, każda aplikacja działa we własnym procesie oraz posiada swój własny unikatowy stos Android Runtime (ART – uruchamia wiele wirtualnych maszyn na urządzeniach o małej pamięci).

Najważniejsze cechy techniki ART:

- Zoptymalizowane garbage collector (usuwanie z pamięci nieużywanych obiektów).
- Kompilację w czasie rzeczywistym.
- Lepsza obsługa debugowania – szczegółowe wątki diagnostyczne i raportowanie awarii.

Java Freamwork API

Interfejs graficzny systemu operacyjnego Android jest dostępny dzięki interfejsowi API napisanego w języku Java. Interfejsy te tworzą bloki niezbędne do uruchomienia aplikacji na Androida, jak najbardziej upraszczając wykorzystanie podstawowych modułowych komponentów systemu i usług.

2.3. Cykl Życia Aplikacji

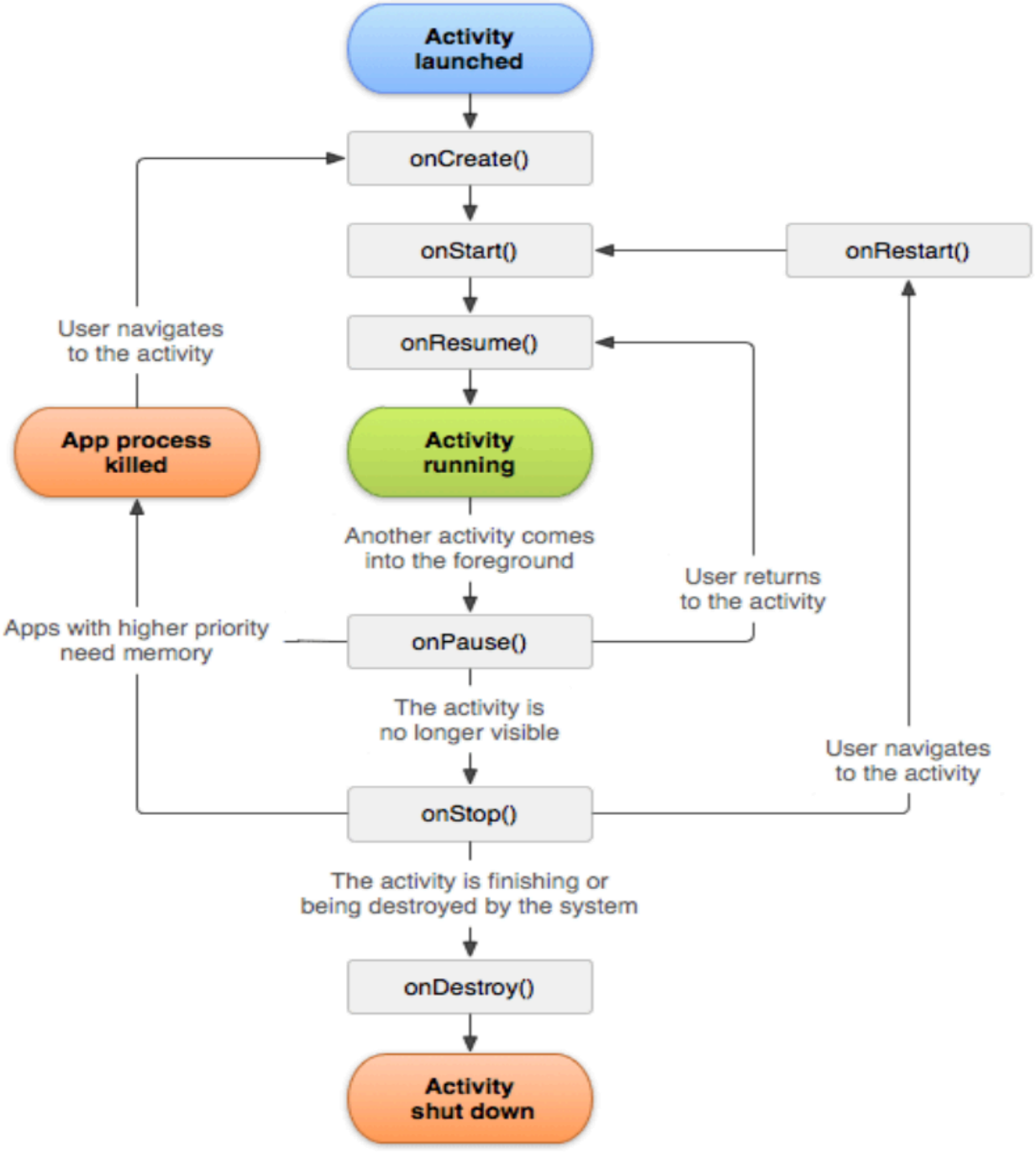
Aktywność w aplikacji Android posiada cztery stany w którym może się znajdować:

- Jest główną aplikacją wyświetlaną na pierwszym planie.
- Jest wyświetlana na ekranie, ale nie jako pierwszoplanowa – wyświetlane jest okienko, które przysłania je w jakiejś części.
- Jest wyłączona – brak jej w pamięci operacyjnej.
- Jest wstrzymana – jest zminimalizowana i niewidoczna na ekranie głównym. Działa w tle. Dzieje się tak, gdy przechodzimy z jednej aktywności do drugiej. Aktywność będąca w tym stanie może zostać w każdej chwili wyłączona całkowicie, w razie zapotrzebowania na większą ilość pamięci RAM.

Dla każdego stanu w aplikacji Android wywoływana jest odpowiednia metoda, która jest uruchamiana przy przechodzeniu aktywności między stanami:

- **onCreate ()** – metoda wywoływana w momencie utworzenia naszej aktywności.
- **onStop ()** – wywoływana, gdy aplikacja jest zamykana.

- **onPause ()** – uruchamia się w momencie zastopowania działania aplikacji, np. gdy minimalizujemy działającą aplikację, lub wywołujemy obrócenie ekranu.
- **onResume ()** – w momencie przywrócenia aplikacji na ekran startowy, chwilowego wejścia do menu i również przy obrocie ekranu, zostaje wywołana funkcja onResume.



Rysunek 2.2. Grafika przedstawiająca cykl działania aplikacji Android, źródło: https://androiddlaprogramistow.wordpress.com/2013/11/12/cykl-zycia-aktywnosci-activity-lifecycle/

3. OPIS PROTOKOŁÓW KOMUNIKACYJNYCH

3.1. SSH – BEZPIECZNA KOMUNIKACJA MIĘDZY URZĄDZENIAMI

SSH (z ang. Secure Shell – powłoka ochronna) – jest to protokół komunikacyjny wykorzystywany w sieci komputerowej TCP/IP, dzięki któremu użytkownik jest w stanie pracować w terminalu lub powłoce zdalnego systemu, działając jakby to był system lokalny. Secure Shell powstał jako rozwiązanie problemów z brakiem bezpieczeństwa protokołu Telnet. Dostarczyło to możliwość bezpiecznej pracy w konsoli/terminalu zdalnego systemu. SSH funkcjonuje w architekturze klient-serwer, a domyślnym portem wykorzystywanym w tym protokole komunikacyjnym jest port 22.

Protokołem bezpieczeństwa wykorzystanym w SSH jest szyfrowanie za pomocą kryptografii symetrycznej z kluczem jawnym dla celów przesyłania klucza sesji. Najczęściej stosowany jest on przez administratorów do obsługi serwerów, które mogą znajdować się nawet w różnych miejscach geograficznych. Wykorzystują go również osoby, które mają np. wykupione serwer VPS i dzięki SSH nimi sterują. Protokół ten wywodzi się od podobnego protokołu zdalnego dostępu TELNET, lecz TELNET (najstarszy protokół warstwy aplikacji) nie szyfruje komunikacji pomiędzy klientem a serwerem, a tekst jest wysyłany jawnym tekstem, więc jest możliwość przejęcia komunikacji.

3.2. ALGORYTMY SZYFROWANIA KOMUNIKACJI SSH

Domyślnym algorytmem szyfrowania komunikacji protokołu SSH jest algorytm RSA (możliwe jest również wykorzystanie szyfrowania danych za pomocą nieco słabszego algorytmu DSA).
Algorytm RSA jest jednym z najbardziej popularnym asymetrycznym algorytmem kryptograficznym. Opracowany przez Rona Rivesta, Leonarda Adelmana i Adi Shamira w 1977 roku. Jest to pierwszy algorytm, który można stosować zarówno do szyfrowania jak i podpisów cyfrowych.
RSA posiada 2 typy kluczy – klucz prywatny oraz klucz publiczny. Klucz publiczny umożliwia jedynie zaszyfrowanie danych i w żaden sposób nie ułatwia ich odczytania, więc nie musi być w żaden sposób chroniony. Klucz prywatny jest przechowywany pod nadzorem i służy do odczytywania zakodowanych informacji za pomocą klucza publicznego. Również zaszyfrowanie wiadomości za pomocą klucza publicznego jest możliwe i dzięki temu RSA może zostać wykorzystany do cyfrowego podpisywania dokumentów.
System RSA daje możliwość bezpiecznego przesyłania danych w środowisku, w którym może dojść do próby wykradnięcia danych. Bezpieczeństwo szyfrowania opiera się w tym algorytmie na bardzo czasochłonnym obliczeniowo faktoryzacji (znajdowaniu czynników pierwszych) dużych liczb złożonych. Obecnie nawet najszybszym komputerom może to zająć wiele dziesiątków lat, dlatego algorytm ten jest bezpiecznym dla łącza chociażby właśnie protokołu SSH.

3.3. PROTOKÓŁ KOMUNIKACYJNY HTTP

HTTP (ang. Hypertext Transfer Protocol) jest protokołem aplikacyjnym, który przesyła dokumenty hipertekstowe do protokołu sieci WWW (ang. World Wide Web). Funkcją protokołu HTTP jest protokół „request-response" (ang. zapytaj – odpowiedz) w komunikacji klient – serwer. Klient wysyła żądanie, gdzie każde żądanie powiązane jest z zasobem. Zasobem może być zarówno strona HTML, plik z kodem JavaScript lub zdjęcie. Sam protokół HTTP nie jest w stanie określić czym dokładnie jest dany zasób, a określa jedynie ścieżkę do danego zasobu. Każdy zasób ma swój unikatowy identyfikator – URI (ang. Uniform Resource Identifier).
Podzbiorem URI jest URL (ang. Uniform Resource Locator), w czym URI można traktować jako zbiór znaków, który daje możliwość unikalne identyfikować zasób, a URL zawiera informację dotyczące „położenia" danego zasobu w serwerze.
Poszczególnymi częściami URL jest:

- Scheme – określa protokół komunikacyjny. Najczęściej jest nim HTTP lub HTTPS. HTTPS jest rozszerzeniem protokołu HTTP, gdzie komunikacja między serwerem a klientem jest szyfrowana.
- Host – dla protokołu HTTP jest to po prostu nazwa domeny internetowej lub adresu IP. Domena może mieć nazwę www.stronainternetowa.pl, a przykładowy adres IP: 192.168.0.2.
- Port – jest to numer wykorzystany przez serwer. Serwer działa na określonym porcie i na nim przekazuje wszystkie informacje. Popularne protokoły jak http, https czy np. ssh mają swoje standardowe numery portów. Protokół http używa portu 80, https 443, a ssh 22.

4. KAMERA WYKORZYSTANA AKWIZYCJI DO OBRAZU

4.1. OPIS KAMERY

Kamera użyta w projekcie to „Camera HD D”, która charakteryzuje się tym, że jest to moduł 5-megapikselowej kamery z sensorem OV5647 przeznaczonej dla platformy Raspberry Pi. Współpracuje ona ze wszystkimi wersjami Raspberry Pi (Raspberry Pi 3 model B, Raspberry Pi 3 model B+, Raspberry Pi 2 model B, Raspberry Pi 1 model B+).

Właściwości:

- Matryca OV5647, 5 megapikseli
- Wielkość matrycy: ¼ cala
- Apertura: 2.8
- Ogniskowa: 3,37 mm
- Kąt widzenia: 72,4 stopni
- Maksymalna rozdzielczość sensora: 1080p
- Maksymalna rozdzielczość zdjęć 2592 x 1944
- Format filmów: 1080p30, 720p60, 640 x 480p 60/90
- Wielkość: 25 x 24 x 9 mm

W zakupionym zestawie znajduje się moduł z kamerą oraz taśma połączeniowa. Większość kamerek w tego typu sprzętach używa interfejsu USB, który jest jednak stosunkowo powolny, więc przesyłanie wysokiej jakości filmów może powodować pewne problemy. Jednocześnie kamery korzystające z USB powodują bardzo duże wykorzystanie pasma tego interfejsu, co może za bardzo obciążać magistrale i utrudniać pracę innych połączonych urządzeń. Dlatego też moduł Raspberry Pi Camera HD D korzysta z interfejsu CSI (ang. Camera Serial Interface), który jest dedykowany właśnie do kamer. Takie rozwiązanie pozwala na mniejsze obciążenie magistrali USB, a dodatkowo interfejs kamery jest wspierany przez akcelerator graficzny (GPU) wbudowanego w Raspberry Pi.

4.2. PODŁĄCZENIE KAMERY

Podłączenie kamery ogranicza się jedynie do wpięcia taśmy:

- Na Raspberry Pi taśmę wpinamy w złącze opisane jako CAMERA. Taśma musi być skierowana srebrnymi stykami w stronę złącza HDMI.
- Na kamerze taśmę wpinamy, aby srebrne styki były skierowane w stronę obiektywu.

W momencie wpinania taśmy trzeba podnieść blokadę złącza, następnie wsunąć taśmę i docisnąć blokadę. Należy jednocześnie uważać, żeby nie wpinać kamerki do złącza z zaciśniętą blokadą, gdyż może to spowodować uszkodzenie mechaniczne. W momencie wpięcia kamery zostaje on w jednej pozycji filmowania, lecz obraz przesłany przez kamerę można w dowolny sposób obracać programowo.

5. AUTONOMIA W ROBOTYCE

5.1. CO TO JEST AUTONOMIA

Autonomia – techniki sztucznej inteligencji, takie jak planowanie, tworzenie harmonogramów i systemów wieloagentowych. W ogólnym rozumieniu autonomiczność rozumiana jest jako umiejętność do wykonywania/rozwiązywania przez robota określonych i zdefiniowanych zadań w otoczeniu rzeczywistym, przy czym nie oddziaływają na niego żadne urządzenia peryferyjne. Wszystkie decyzje dotyczące sterowania oraz przemieszczania są podejmowane wyłącznie w obrębie platformy robota, który wykorzystuje do tego własne struktury mechaniczne, elektroniczne oraz programowe. Stopień autonomiczności zdeterminowany jest przede wszystkim poprzez:

- Algorytmy odpowiadające za sterowanie robota, postrzegania środowiska za pomocą danych z czujników/kamer/akcelerometrów itp., w jakim znajduje się robot oraz adaptacji do niego.
- Właściwości sprzętowe, znane jako stopień zaawansowania układów służących do pomiarów, określonych warunków panujących w otoczeniu robota, oraz mocy i wydajności jednostek obliczeniowych oraz sposoby interpretacji i weryfikacji sygnałów docierających do konkretnych aparatów badających otoczenie.

Roboty, które zostały pozbawione autonomiczności, dokonują przemieszczenia za pomocą instrukcji zadanych przez człowieka w czasie rzeczywistym. Tego typu roboty to roboty teleoperowane. Nie posiadają one bezpośredniego funkcjonowania poprzez komunikację człowieka z maszyną, lecz za pomocą interfejsu HRI (z ang. interfejs człowiek-robot). Roboty przeciwne do robotów teleoperowanych to roboty w pełni autonomiczne, lecz częściej roboty te są w jakimś stopniu autonomiczne. Roboty w jakiejś części autonomiczne są nazywane pośrednimi, czyli takimi, gdzie w wykonywanie poszczególnych zadań bierze udział operator, który odpowiada za podjęcie decyzji, które wymagają zbyt dużej złożoności obliczeniowej. Podejmowanie odpowiednich decyzji należy jednak do maszyny, która realizuje zamierzony cel w fazach w których za nadzorowanie poszczególnych kroków wykonawczych nie bierze udział operator.

5.2. PODZIAŁ ROBOTÓW ZE WZGLĘDU NA SPOSÓB PROGRAMOWANIA I MOŻLIWOŚCI KOUNIKACYJNE

5.2.1. GENERACJA I

Roboty I generacji (nauczane) są to urządzenia, które zostały posiadają pamięć, w której zostały zaimplementowane rozkazy i w momencie uruchomienia urządzenia zdolne są one do wykonywania zaprogramowanych czynności bez ingerencji człowieka. Roboty I generacji nie są więc zdolne do samodzielnego zbierania informacji o środowisku, w którym się porusza oraz mają ograniczone właściwości funkcyjne. Do tego typu robotów można zaliczyć np.: roboty przemysłowe, które są zaprogramowane na przenoszenie obiektów, tylko w momencie, gdy obiekt ten znajduje się w tym samym miejscu i w odpowiedniej rotacji.

5.2.2. GENERACJA II

Roboty II generacji (uczące się) są to urządzenia, które potrafią rozpoznać konkretny obiekt zawarty w zbiorze obiektów, bez względu na jego położenie czy kształt. Daje to możliwość reagowania robota nawet po zmianie jego miejsca pracy względem poszukiwanego obiektu. Roboty II generacji mają jednak ograniczoną możliwość, jeśli chodzi o rozróżnianie kształtów i położeń obiektów, dzięki wykorzystanym w nich układów czujników oraz różnego typu systemów wizyjnych połączonych z jednostką komputerową, która bada i przetwarza sygnały.

5.2.3. GENERACJA III

Roboty III generacji (inteligentne) to urządzenia, które zostały na starcie zaopatrzone w sztuczną inteligencję, która charakteryzuje się podejmowaniem decyzji, ze względu na działanie zmieniających się warunków oraz pracy w nieznanym sobie otoczeniu. Takie roboty zazwyczaj również wyposażone są w komplety czujników, które są odpowiedzialne za ingerencję zewnętrznych aspektów na ruch robota.

5.3. PODZIAŁ ROBOTÓW ZE WZLĘDU NA UKŁAD NAPĘDOWY

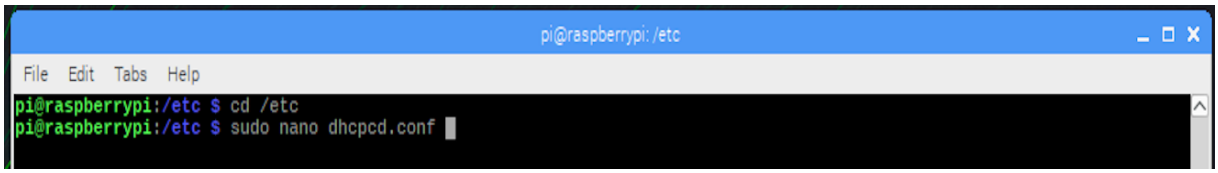
- Roboty z napędem elektrycznym
- Roboty z napędem pneumatycznym
- Roboty z napędem hydraulicznym
- Roboty z napędem mieszanym

6. KONFIGURACJA RASPBERRY PI

6.1. USTAWIENIE STATYCZNEGO IP

Ustawienie statycznego IP w platformie mobilnej jest wygodnym rozwiązaniem dla programowania aplikacji dedykowanej na tę platformę. Pozwala to programiście zaimplementować w aplikacji jedną wartość adresu IP, po którym łączy się z robotem, gdyż w momencie połączenia urządzenia do innej sieci i nadaniu innego adresu IP, należałoby za każdym razem zmieniać te dane w aplikacji, co byłoby niezwykle uciążliwe. Statyczny adres IP to adres IP, który został ręcznie skonfigurowany dla konkretnego urządzenia, w porównaniu do urządzenia, któremu takowy adres został przypisany przez DHCP serwer.
Przeciwieństwem statycznego adresu IP jest zawsze zmieniający się dynamiczny adres IP, który nie jest na stałe przypisany do danego urządzenia. Jest on używany przez dany czas, po którym wraca do puli adresów, dając możliwość innym urządzeniom do jego wykorzystania.
Główną wadą statycznego IP jest to, że należy samemu je ustawiać ręcznie, co na pewno wymaga więcej pracy niż po prostu podłączając router i dać możliwość przydzielania dynamicznych adresów IP przez DHCP. Kolejną wadą jest to, że wpływa to na zabezpieczenie sieci, gdyż używanie przez dłuższy czas jednego adresu IP, daje hackerom większą ilość czasu, aby znaleźć luki w urządzeniach. Mimo wszystko w określonych przypadkach jak do łączenia się z robotem, który w przyszłości będzie korzystał z różnych sieci, ustawienie statycznego IP, jest wręcz obowiązkowym rozwiązaniem.

Aby ustawić statyczny adres IP w Raspberry Pi 3 należy wejść do katalogu: ‘/etc’, a następnie edytować plik: ‘dhcpcd.conf’:



Rysunek 6.1. Komenda do wejścia pliku konfiguracyjnego.

W pliku tym na samym dole należy dodać odpowiednie linijki tekstu:

Rysunek 6.2. Edytowane wnętrze pliku konfiguracyjnego

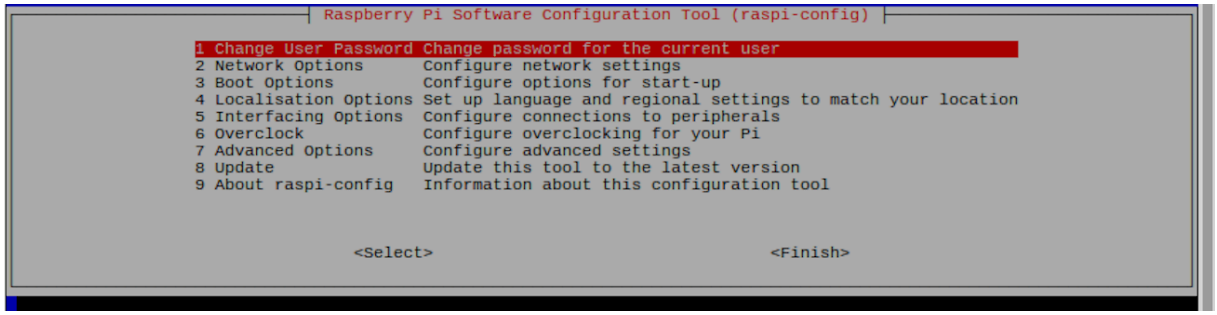
Dzięki tej zmianie zarówno w momencie podłączenia urządzenia przez ethernet jak i po sieci bezprzewodowej WLAN, adres pozostaje stały, w tym przypadku adres IP to: „192.168.0.2”. Nie używamy adresu „192.168.0.1”, gdyż został on zarezerwowany dla routera. Adres statyczny może być w takim razie jakikolwiek adres z zakresu 192.168.0.2 – 192.168.0.255.

6.2. AKTYWOWANIE KAMERY ORAZ ŁĄCZA SSH

Program do kontroli platform mobilnej opiera się na komunikacji za pomocą protokołu SSH telefonu mobilnego z platformą. Do wyświetlania obrazu na żywo użyta została kamera, którą zarówno jak możliwość łączenia się przez SSH należy na samym początku projektu ustawić w konfiguracji Raspberry Pi. W pierwszej kolejności odpalić należy terminal i wpisać komendę: „sudo raspi-config”.

Rysunek 6.3. Komenda aktywująca okienko do konfiguracji w Raspberry Pi

W następnej kolejności na ekranie pojawia się ekran konfiguracji podstawowych narzędzi Raspberry Pi:

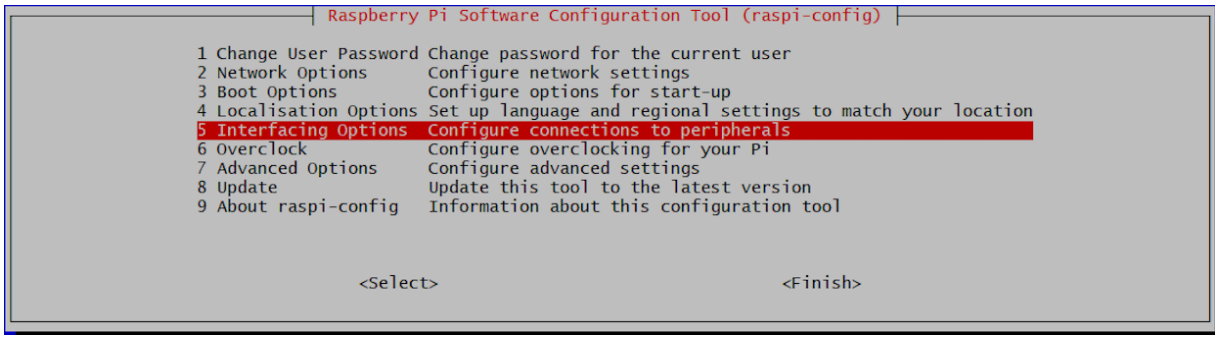


Rysunek 6.4. Okienko do konfiguracji w Raspberry Pi

W okienku tym możemy zaobserwować możliwe zmiany jakie możemy zastosować w naszym urządzeniu:

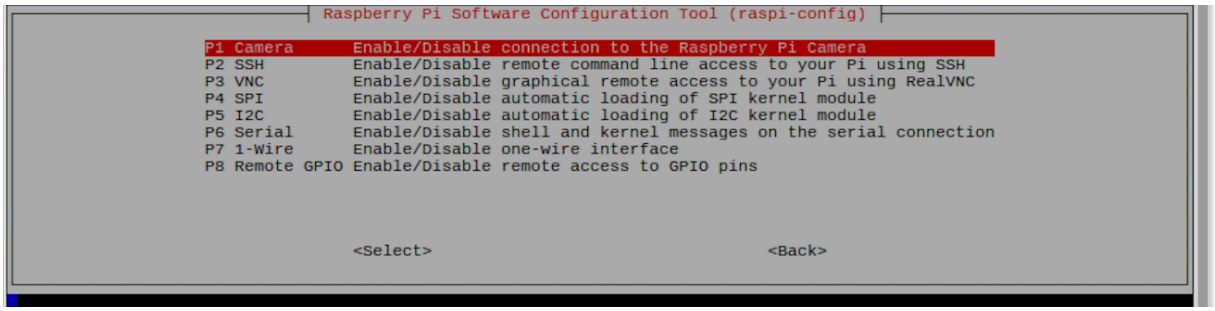
1. Zmiana hasła użytkownika konta, na którym aktualnie się znajdujemy.
2. Konfiguracja ustawień sieci.
3. Konfiguracja ustawień w momencie uruchomienia Raspberry Pi.
4. Ustawienie języka oraz czasu urządzenia względem lokalizacji.
5. Ustawienia łączenia urządzeń peryferyjnych.
6. Ustawianie wydajności obliczeniowej urządzenia.
7. Ustawienia zaawansowane typu: konfiguracja audio, czy przydzielania pamięci.
8. Aktualizowanie urządzenia.
9. Informacje o oknie konfiguracyjnym.

Aby włączyć możliwość łączenia się przez SSH oraz aby aktywować kamerę należy wejść opcję numer 5, czyli „Interfacing Options”:



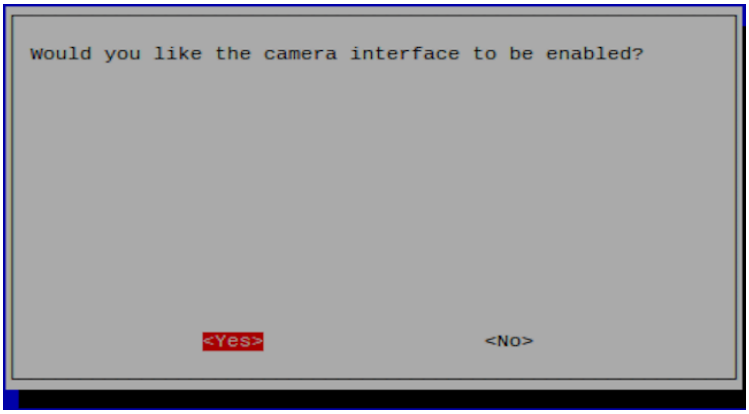
Rysunek 6.5. Opcja do zmiany ustawień urządzeń peryferyjnych

Następnie wybieramy odpowiedni moduł, który chcemy aktywować:



Rysunek 6.6. Wybrany moduł do aktywacji kamery

Po zatwierdzeniu dostajemy informację czy dany moduł ma zostać włączony:



Rysunek 6.7. Potwierdzenie aktywowanie kamery

Na koniec należy powtórzyć wszystkie kroki dla włączenia SSH, zresetować urządzenie i w tym momencie oba moduły zostaną aktywowane w Raspberry Pi.

7. REALIZACJA PROJEKTU

7.1. OGÓLNY ZAMYSŁ APLIKACJI

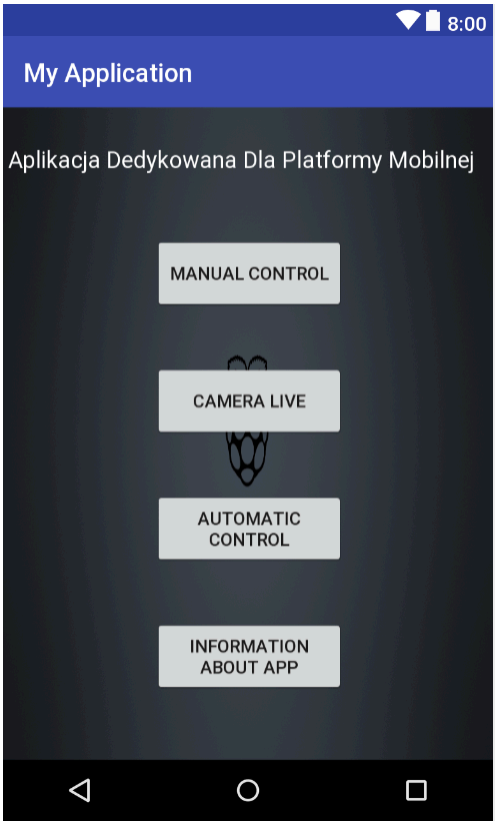
Głównym założeniem projektu, było umożliwienie wykorzystania w aplikacji przez klienta trzech podstawowych funkcji:

1. Możliwość sterowania manualnego
2. Możliwość jazdy autonomicznej.
3. Przesłanie obrazu na żywo z robota do aplikacji.

7.2. AKTYWNOŚĆ PIERWSZA - MENU

7.2.1. REPREZENTACJA GRAFICZNA

Na samym początku został utworzony projekt w Studio Android, gdzie główne okno zostało zaimplementowane jako proste menu z którego możemy przenieść się do odpowiednich funkcji projektu.



Rysunek 7.1. Reprezentacja graficzna menu w aplikacji

Menu graficzne składa się z 2 podstawowych elementów interfejsu użytkownika wykorzystywanego w programowaniu w Android Studio: TextView oraz Button (ang. pole tekstowe, przycisk). Pole tekstowe jest wyłącznie z informacją jakiego typu aplikacją

użytkownik ma styczność. Następnie 4 przyciski umożliwiające przeniesienie się do innych aktywności odpowiadających za najważniejsze funkcje programu.
Jest również możliwość odpalenia interfejsu graficznego w trybie tekstowym, gdzie ukazane są wszystkie jego atrybuty:

```
<TextView
    android:id="@+id/textView3"
    android:layout_width="407dp"
    android:layout_height="39dp"
    android:layout_marginLeft="4dp"
    android:layout_marginStart="4dp"
    android:layout_marginTop="28dp"
    android:text="Aplikacja Dedykowana Dla Platformy Mobilnej"
    android:textColor="@android:color/white"
    android:textSize="18sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/secondActivity"
    android:layout_width="150dp"
    android:layout_height="60dp"
    android:layout_marginEnd="116dp"
    android:layout_marginRight="116dp"
    android:layout_marginTop="100dp"
    android:text="Manual Control"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Rysunek 7.2. Wnętrze kodu odpowiedzialnego za interfejs graficzny

W atrybucie „TextView” możemy zaobserwować podstawowe informacje o danym elemencie:

1. id – specyficzna nazwa elementu, dzięki któremu będziemy mogli obsługiwać go i zmieniać w kodzie Java.
2. layout-width – szerokość elementu
3. layout-height – wysokość elementu
4. layout-margiLeft – odległość od lewej krawędzi ekranu
5. layout-margiStart – odległość od lewej krawędzi ekranu
6. layout-margiTop – odległość od górnej krawędzi ekranu
7. text – tekst wyświetlany w odpowiednim miejscu w atrybucie
8. textColor – kolor tekstu
9. textSize – rozmiar tekstu
10. layout-constraintEnd-toEndOf – dany atrybut jest połączony od lewej strony z krawędzią ekranu (możliwe wykorzystanie danej funkcji w Constraint Layout).
11. layout-constraintTop-toTopOf – dany atrybut jest połączony od lewej strony z krawędzią ekranu

Zarówno interfejs graficzny, gdzie możemy ustawiać ręcznie odpowiednie elementy, jak również tekstowa reprezentacja danych atrybutów daje możliwość w zależności od programisty bardziej lub mniej wygodnego oraz precyzyjnego budowania interfejsu, który jest pierwszym elementem, jaki widzi użytkownik i od którego zależy wygoda korzystania z aplikacji.

7.2.2. ANDROID MANIFEST – CHARAKTERYSTYKA APLIKACJI

AndroidManifest.xml jest plikiem, w którym opisana została charakterystyka podstawowa aplikacji oraz znajdują się w niej definicje podstawowych komponentów. Znajdują się w niej takie informacje jak typ systemu kodowania Unicode, nazwę „package" gdzie podana jest lokalizacja aplikacji i np. uprawnienia dotyczące aplikacji.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.praca.myapplication">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".Main2Activity" />
        <activity
            android:name=".Main3Activity"
            android:label="Main3Activity"
            android:theme="@style/AppTheme.NoActionBar" />
        <activity android:name=".information" />
        <activity android:name=".automatic"></activity>
    </application>

</manifest>
```

Rysunek 7.3. Kod Android Manifest

Budowa AndroidManifest.xml

- Jak widać na rysunku 7.3 w stworzonej aplikacji do sterowania robotem mobilnym został użyty system kodowania „utf-8” (system kodowania Unicode, który używa od 1 do 4 bajtów w celu zakodowania znaku).
- „uses-permission …” służy do określenia uprawnień aplikacji, która potrzebuje odpowiednich żądań, by używać wrażliwych danych użytkownika jak dostęp do kontaktów czy smsów, oraz niektórych funkcji systemu typu: dostęp do kamery, czy zgodę na wykorzystanie Internetu. Każde uprawnienie jest dodawane w unikalnie zidentyfikowanej komendzie. W aplikacji zostały wykorzystane dwie komendy przydzielenia uprawnień:
  - „android. permission. INTERNET” – zezwala aplikacji na otworzenie połączenia internetowego (wykorzystanego przez łącze SSH oraz Sockety).
  - „android. permission. ACCESS_NETWORK_STATE” – zezwala aplikacji na pobieranie informacji o sieci.
- Opis komponentów aplikacji, w czym zawarte się wszystkie utworzone aktywności oraz informacje o nadawcy i dostawcy treści aplikacji. Wszystkie komponenty muszą definiować podstawowe własności takie jak nazwy utworzonych klas Java.

W programie do budowy aplikacji Android Studio, plik manifest jest stworzony automatycznie przy tworzeniu projektu i w większości przypadków elementy dodane przy rozszerzeniu aplikacji są automatycznie dodawane do pliku.

7.2.3. PLIK GŁÓWNY – JAVA

Kod programu do obsługi aplikacji na platformę Android jest napisany w języku Java. Jest to podstawowy język do pisania aplikacji na tę platformę (w ostatnim czasie na popularności zyskuje Kotlin), który cechuje się interesującymi aspektami:
- Jest to język obiektowy, który opiera się na tworzeniu aplikacji w taki sposób, aby w jak najlepszy sposób odzwierciedlał otaczający nas świat, dzięki czemu jest bardzo intuicyjnym językiem.
- Posiada wiele gotowych bibliotek, które bardzo ułatwiają pracę programiście.

Aktywność pierwsza jest to tylko menu z którego możemy dostać się innych aktywności, w których zawarty jest kod odpowiadający za podstawowe funkcje programu. Dzięki 4 przyciskom dodanym w pliku xml możemy wywoływać funkcje odpowiadające za to przejście.

W momencie utworzenia nowego projektu na ekranie w pliku Java pojawia się:

```
package com.example.praca.testproject;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Rysunek 7.4. Plik Java po utworzeniu nowego projektu w Android Studio

Na samej górze pojawia się tzw. „Package", który zawiera informacje o implementacji i specyfikacji Java i posiada unikatową nazwę w celu zapobiegania konfliktu nazw, dlatego jeden projekt posiada tą samą nazwę „package" we wszystkich aktywnościach.
Następnie pojawiają się słowa kluczowe „import" po których jest nazwa biblioteki, która została dołączona do projektu, dzięki czemu możemy korzystać z odpowiednich gotowych funkcji zaimplementowanych w danej bibliotece.
Pod kodem dodającym biblioteki znajduje się główna klasa danej aktywności. W nim znajduje się cały kod odpowiadający za program działający w jej obrębie. Klasa ta dziedziczy po podstawowej klasie dla aktywności, która wspiera bibliotekę odpowiadającej za aktywność. W klasie tej została utworzona metoda „onCreate", która przysłania klasę z dziedziczącej klasy „AppCompatActivity", dzięki czemu dajemy informację do Dalvik VM do uruchomienia kodu znajdującego się w nim, dodatkowo do metody „onCreate" klasy rodzica. Bez dodania wywołania metody przez słowo kluczowe „super" spowoduje, że kod w aktywności nigdy się nie uruchomi.
Funkcja „setContentView" pozwala powiązać się aktywności z odpowiednim plikiem projektu graficznego. R oznacza „Resource" czyli źródło, a layout dany model, do którego się łączymy.

W aktywności menu dodatkowo dołączone są do podstawowego kodu, odpowiednie funkcje od przypisania zmiennych do obsługi przycisków oraz zmiany aktywności na inne.

```
Button manualControl = (Button) findViewById(R.id.secondActivity);
Button cameraLive = (Button) findViewById(R.id.camera);
Button automaticControl = (Button) findViewById(R.id.automatic);
Button inform = (Button) findViewById(R.id.info);
```

Rysunek 7.5. Przypisanie w pliku Java odpowiednich komponentów z pliku xml

Aby przypisać odpowiedni przycisk do zmiennej w kodzie, należy najpierw utworzyć zmienną o odpowiednim typie (w tym przypadku „Button"), a następnie znaleźć go po i przypisać do tej zmiennej po nazwie „id". Funkcja „findViewById" znajduje w kodzie xml

odpowiedni przycisk, do którego ścieżkę musimy podać w argumentach funkcji. Następnie możemy do tej zmiennej przypisywać odpowiednie akcje w programie.

```
manualControl.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent i = new Intent( packageContext: MainActivity.this, Main2Activity.class);
        startActivity(i);
    }
});
```

Rysunek 7.6. Przypisanie funkcji do kontroli przycisku w aplikacji

Podstawową funkcją w Android Studio do obsługi przycisków, jest metoda „setOnClickListener", która wywołuje odpowiednią akcję, zawartą w kodzie w ciele tej funkcji w momencie przyciśnięcia przycisku. Aby przejść do innej aktywności należy użyć intencji. Są to obok aktywności jedne z podstawowych komponentów, z których składa się Androidowa aplikacja. Odpowiada od przede wszystkim za obsługę rozkazów, które zostały wydane przez użytkownika. Za jej pomocą można wprowadzić komunikację pomiędzy aplikacjami (lub mniejszymi komponentami typu: Aktywności czy Usługi), lecz najważniejszą rolą intencji jest uruchamianie odpowiednich aplikacji/Aktywności.
Intent można uruchomić na dwa sposoby – jawnie oraz niejawnie. Jawne wywołanie to takie, gdzie został jasno sprecyzowany obiekt, który chcemy stworzyć. W tym przypadku jednym z argumentów konstruktora Intencji zostaje obiekt typu „Class" (w przypadku mojej aplikacji „Main2Activity.class" odpowiadającą za sterowanie manualne), który wskazuje na klasę, której obiekt chcemy stworzyć/uruchomić, a następnie za pomocą „startActivity" uruchamiamy daną aktywność.

7.3. AKTYWNOŚĆ DRUGA – STEROWANIE MANUALNE.

7.3.1. REPREZENTACJA GRAFICZNA

Aktywność odpowiedzialna za sterowania manualne składa się z pięciu komponentów typu „Button” oraz jednego typu „TextView”. Każdy przycisk odpowiada za wykonanie założonej funkcji.

Rysunek 7.7. Reprezentacja graficzna sterowania manualnego

7.3.2. PLIK GŁÓWNY – JAVA

W kodzie głównym została zawarta jedna funkcja łączenia po protokole SSH pomiędzy Raspberry Pi, a telefonem. Do łączenia się po SSH należało najpierw dodać odpowiednią bibliotekę w Android Studio. Zdecydowałem się na bibliotekę JSCH (Java Secure Channel), która jest popularną biblioteką oraz posiada szeroką dokumentację dotyczącej jej wykorzystania. Na oficjalnej stronie biblioteki zostały zawarte informacje, że biblioteka ta zezwala na łączność po protokole SSH, użycia przekierowania portów oraz przesyłania plików. Do sterowania manualnego wykorzystano 5 przycisków, przy którym każdy przycisk łączy się po SSH oraz uruchamia odpowiedni skrypt na platformie mobilnej.

```
public static void steruj(String host, String user, String password, String command1) throws Exception {

    try {

        java.util.Properties config = new java.util.Properties();
        config.put("StrictHostKeyChecking", "no");
        JSch jsch = new JSch();
        Session session = jsch.getSession(user, host,  port: 22);
        session.setPassword(password);
        session.setConfig(config);
        session.connect();

        Channel channel = session.openChannel( type: "exec");
        ((ChannelExec) channel).setCommand(command1);

        channel.disconnect();
        session.disconnect();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Rysunek 7.8. Główna funkcja odpowiedzialna za sterowanie manualne.

Funkcja „steruj" przyjmuje 4 argumenty:

- Adres IP urządzenia
- Nazwę użytkownika
- Hasło niezbędne do połączenia się przez protokół SSH
- Komendę, która ma zostać wykonana

Na samym początku funkcji został zawarty kod, który zmienia podstawowe uprawnienia połączenia pomiędzy urządzeniami. Poprzez kod „config.put („StrictHostKeyChecking", „no")" zmieniamy ustawienia w protokole SSH. W tym momencie JSCH automatycznie dodaje nowy klucz hosta do zapisanych w pliku kluczy hosta użytkownika. Jest też możliwość ustawienia wartości na „yes" oraz „ask". Gdyby w miejsce „no" wpisać zmienną „yes", wtedy JSCH nigdy automatycznie nie dodałoby klucza prywatnego hosta i odmówi połączenia się z urządzeniem, które miało zmieniony klucz prywatny. W tym przypadku użytkownik musi manualnie dodać wszystkie nowe hosty.
Jeśli w miejsce „no" wstawimy „ask", nowy klucz prywatny hosta zostanie dodany do pliku znanych hostów wyłącznie wtedy, gdy użytkownik potwierdzi, że wyraża zgodę na dodanie go do pliku, a JSCH odmówi mu dostępu, jeśli połączony host zmieni kiedykolwiek swój klucz prywatny.
W następnej części funkcji zostaje utworzony obiekt JSch, dzięki któremu możemy wykorzystać metody w niej zawarte jak „getSession", które pozwala łączyć się przez protokół podając nazwę użytkownika, numer hosta, oraz port (domyślnie dla protokołu SSH jest to port 22). W „setPassword" ustawione zostaje przekazane przez argument hasło umożliwiające połączenie się między urządzeniami, a następnie za pomocą „session. Connect ()" połączenie to zostaje otwarte.
W dalszej części kodu zostaje utworzony obiekt „Channel" który odpowiada za zdalne wykonywanie programu. Podając argument „exec" w metodzie „openChannel", komendy, które mają się wykonać na zdalnym urządzeniu zostają przesłane jako zmienna string w metodzie „setCommand ()". Serwer SSH przekazuje je jednocześnie do powłoki systemowej,

gdzie zostają wykonane. Będą one wykonywane, dopóki powłoka systemowa z jakiegoś powodu nie zostanie zamknięta.
Na koniec funkcji zostaje zamknięty „channel” oraz „session”, aby komunikacja między urządzeniami została zakończona.
Aby wywołać odpowiedni skrypt trzeba przypisać do każdego przycisku odpowiednią akcję i wykonanie odpowiedniego skryptu.

```
Button forward1 = (Button) findViewById(R.id.forward);
forward1.setOnClickListener((v) → {
        new AsyncTask<Integer, Void, Void>() {
            @Override
            protected Void doInBackground(Integer... params) {
                try {
                    steruj( host: "192.168.0.2",  user: "pi",  password: "raspberry",  command1: "python Desktop/Adafruit_MotorHAT/forward.py");
                } catch (Exception e) {
                    e.printStackTrace();
                }
                return null;
            }
        }.execute(1);
});
```

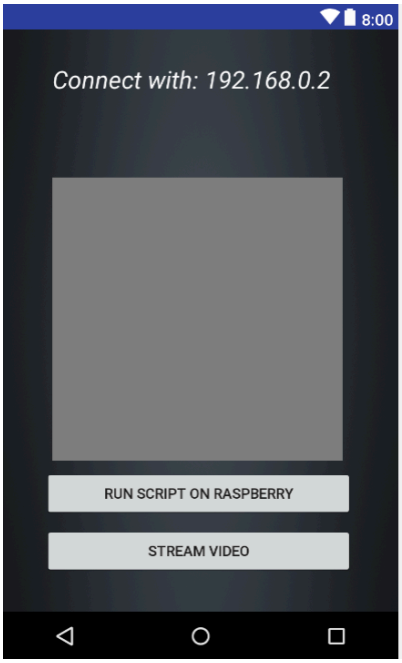
Rysunek 7.9. Kod funkcji do sterowania manualnego wraz z niezbędnymi argumentami

W momencie utworzenia obiektu „buton” i przypisaniu do niego odpowiedniego przycisku z pliku xml, można przypisać do niego odpowiednią akcję. Przycisk tak jak w przypadku menu zostaje wykorzystany poprzez metodę „setOnClickListener”, lecz w środku oprócz wywołania funkcji do sterowania manualnego znajduje się tzw. „AsyncTask”. W aplikacjach czy programach często dochodzi do sytuacji, kiedy musimy wykonać operację, która zajmuje więcej czasu. Tak samo w naszym przypadku połączenie się po protokole SSH może zająć bardzo krótki czas lub trochę dłużej, gdy połączenie internetowe jest niestabilne, więc AsyncTask daje nam w tym przypadku możliwość uruchomienia kodu programu w tle. Definiowany jest on przez 3 typy generyczne: Parametry, Progres oraz Wynik. Następnie należy wywołać metodę „doInBackground”, aby dany kod mógł wykonywać się w tle.
Dla bezpieczeństwa kod został zawarty w blokach wyjątku try i catch. W momencie jakiegoś błędu wywołanego w kodzie pod słowem kluczowym „try”, program automatycznie wychwytuje ten błąd i zostaje on wyświetlony na konsoli.
Funkcja steruj przyjmuje w tym przypadku 4 argumenty, które odnoszą się do konkretnej platformy mobilnej. Adres urządzenia jest stały: „192.168.0.2”, nazwa użytkownika „pi”, hasło: „raspberry”, a komenda, która ma się wykonywać na tym urządzeniu to „python Desktop/Adafruit_MotorHat/forward.py”. Jest to standardowe wywołanie programu napisanego w języku python, do którego ścieżka została dokładnie zapisana w tym argumencie. A więc każdy przycisk ma dokładnie takie samo wywołanie akcji po przyciśnięciu przycisku, lecz z innym argumentem wywołania programu znajdującego się w Raspberry Pi. Przycisk forward wywołuje skrypt od jazdy wprzód, backward w tym, left w lewo, right w prawo, a stop powoduje zatrzymanie się wszystkich silników. Program ten działa z lekkim opóźnieniem, lecz nie wpływa ono, aż tak negatywnie na jazdę robotem. Opóźnienie to spowodowane jest tym, że za każdym razem aplikacja musi na nowo łączyć się po protokole SSH z urządzeniem, a następnie wywołać odpowiedni skrypt, lecz po licznych testach w warunkach domowych, opóźnienie to trwało maksymalnie 1.5 s.

7.4. AKTYWNOŚĆ TRZECIA – PRZESYŁANIE OBRAZU

7.4.1. REPREZENTACJA GRAFICZNA

Aktywność trzecia składa się z czterech komponentów. Jedno pole TextView, dwóch Button oraz jednego VideoView.



Rysunek 7.10. Reprezentacja graficzna przesyłania obrazu na żywo

Na rysunku 7.10 pokazano aktywność, która posiada wszystkie niezbędne komponenty do wyświetlania obrazu. Przycisk „Run Script On Raspberry" pozwala na wywołanie odpowiedniej komendy na platformie mobilnej, która streamuje obraz z kamery do sieci. Przycisk „Stream Video" pozwala przechwycić ten obraz z sieci i wyświetlić go na komponencie „VideoView". Tekst napisany na samej górze, informuje użytkownika z jakim urządzeniem zostało zawarte połączenie (jego adres IP).

7.4.2. PLIK GŁÓWNY – JAVA

W kodzie głównym zostały zawarte dwie podstawowe funkcje. Jedna z nich jest to dokładnie ta sama funkcja, która znajduje się w aktywności drugiej, odpowiadającej za komunikację między urządzeniami za pomocą protokołu SSH. Druga funkcja jest to kod pozwalający wyświetlić dany obraz na ekranie telefonu.

```
"raspivid -o -t 10000 -w 640 -h 360 -fps 25|cvlc stream:///dev/stdin --sout '#standard{access=http,mux=ts,dst=:8090}' :demux=h264"
```

Rysunek 7.11. Komenda do uruchomienia przesyłania obrazu z Raspberry Pi

Komendą, która zostaje wysłana i uruchomiona na platformie mobilnej, jest komenda znajdująca się na rysunku 7.11 Jest to komenda odpowiadająca za rozpoczęcie przesyłania obrazu z kamery do sieci. W kodzie tym należy ustawić czas przez jaki obraz będzie się

wysyłał. W tym przypadku jest ustawiony na 10 000 s, lecz można go ustawić na dłuższy okres czasu bądź krótszy. Argument -w oraz -h odpowiadają za wysokość (high) oraz szerokość (width) przesyłanego obrazu, a -fps 25 ustawia liczbę kadrów wyświetlanych w ciągu sekundy, czyli częstotliwość, z jaką statycznie obrazy pojawiają się na ekranie. Jest to miara płynności wyświetlania ruchomych obrazów. Do przesyłania obrazu z Raspberry Pi do sieci, należało najpierw zainstalować na nim odpowiedni program umożliwiający takową czynność. W tym przypadku został to program bardzo popularny, czyli „VLC". W kodzie tym widać, że obraz zostaje przesłany po porcie 8090. „Demux=h264" oznacza standard kodowania sekwencji wizyjnych użyty w tym przypadku.

```
private void playStream(){
    Uri UriSrc = Uri.parse("http://192.168.0.2:8090/");
    if(UriSrc == null){
        Toast.makeText( context: Main3Activity.this,
                text: "UriSrc == null", Toast.LENGTH_LONG).show();
    }else{
        streamView.setVideoURI(UriSrc);
        mediaController = new MediaController( context: this);
        streamView.setMediaController(mediaController);
        streamView.start();

        Toast.makeText( context: Main3Activity.this,
                text: "Connect: 192.168.0.2",
                Toast.LENGTH_LONG).show();
    }
}
```

Rysunek 7.12. Funkcja służąca do wyświetlania obrazu

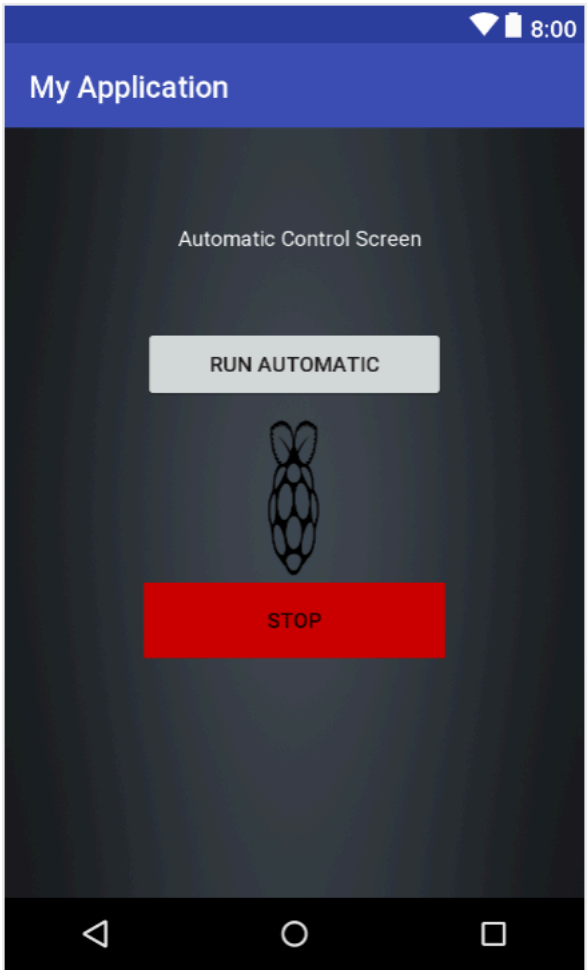
Funkcja użyta do wyświetlania obrazu na ekranie telefonu nie przyjmuje żadnych argumentów. Główną klasą użytą w tej funkcji jest klasa „Uri". Uri – uniform resource identifier to zwarta sekwencja znaków, która identyfikuje abstrakcyjny lub fizyczny zasób. Obiekt Uri używany jest zazwyczaj do identyfikowania obiektu, na którym chcemy pracować, dostarczonym przez referencje. Metoda Uri.parse tworzy nowy obiekt Uri z prawidłowo sformatowanym typem String. W aplikacji do sterowania platformą mobilną obiekt Uri został utworzony jako obiekt odwołujący się do adresu http, na który zostaje przesłany obraz z kamery z Raspberry Pi. Tym adresem jest: http://192.168.0.2:8090/. Adres ten jest adresem IP urządzenia oraz został w nim określony konkretny port, na którym obraz zostaje wysłany. Jeśli do obiektu nie zostaje przypisany żaden adres lub obiekt, na którym chcemy pracować, na ekranie telefonu pojawia się informacja „UriSrc == null", co informuje użytkownika o źle utworzonym obiekcie. W innym przypadku obraz zostaje wyświetlony na ekranie telefonu, jeśli w pierwszej kolejności został wykonany odpowiedni skrypt przesyłający obraz do sieci. Metoda „setMediaController" pozwala na wyznaczenie odpowiedniej instancji, która

umożliwia wyświetlenie sterowania odtwarzanym obrazem na ekranie typu: start, pauza, cofnij. W momencie połączenia aplikacji z adresem, na którym wyświetlany jest obraz na ekranie pojawia się komunikat: „Connect: 192.168.0.2”.

7.5. AKTYWNOŚĆ CZWARTA – AUTONOMIA

7.5.1. REPREZENTACJA GRAFICZNA

Aktywność czwarta składa się z jednego komponentów „TextView” oraz dwóch przycisków. Przyciski odpowiadają za rozpoczęcie pracy autonomicznej oraz jej zakończenie.



Rysunek 7.13. Reprezentacja graficzna automatycznej jazdy platformy mobilnej

7.5.2. PLIK GŁÓWNY – JAVA

Plika główny składa się dwóch klas. Główna klasa, jest to klasa odpowiedzialna za obsługę przycisków, wyświetlania tekstu na ekranie, czy posiada metodę odpowiedzialną za łączenie się po SSH wykorzystaną przez poprzednie aktywności. Składa się jednak jeszcze z dodatkowej klasy, która została stworzona bezpośrednio do łączenia się po gnieździe sieciowym (network socket). W klasie tej zawierają się wszystkie podstawowe metody odpowiadające za komunikację, czyli: połączenie sieciowe, wysyłanie oraz odbieranie wiadomości, zamknięcie połączenia.

KlasaSocketConnect.java:

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class SocketConnect {

    private Socket socket = null;
    private PrintWriter out = null;
    private BufferedReader in = null;
    private byte[] buffer = new byte[157*10];
    InputStream input;
    private String host = null;
    private int port = 7000;

    public SocketConnect(String host, int port) {
        this.host = host;
        this.port = port;
    }

    public void connectWithServer() {
        try {
            if (socket == null) {
                socket = new Socket(this.host, this.port);
                out = new PrintWriter(socket.getOutputStream());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Rysunek 7.14. Klasa SocketConnect.java służąca do komunikacji między Raspberry Pi a Aplikacją

Klasa odpowiedzialna za połączenie między urządzeniami, składa się z odpowiednich bibliotek, które są wykorzystywane do łączenia, oraz przesyłania danych. Konstruktor klasy posiada 2 argumenty, które należy podać w momencie tworzenia obiektu danej klasy. Jest to adres urządzenia oraz port, po którym będziemy komunikować się z urządzeniem. Metoda „connectWithServer" tworzy połączenie między urządzeniami oraz przypisuje do zmiennych odpowiednie metody w celu odczytywania wiadomości oraz ich wysyłania. Funkcja „Socket" po podaniu odpowiedniego adresu oraz portu tworzy połączenie między urządzeniami, jeśli na drugim urządzeniu uruchomiony jest odpowiedni skrypt będący serwerem, który czeka na połączenie się z klientem (w tym przypadku aplikacją).

```java
public void disConnectWithServer() {
    if (socket != null) {
        if (socket.isConnected()) {
            try {
                out.close();
                socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
public void sendDataWithString(String message) {
    if (message != null) {
        out.write(message);
        out.flush();
    }
}
public byte[] receive() throws IOException {
    input = socket.getInputStream();
    input.read(buffer);
    return this.buffer;
}
```

Rysunek 7.15. Metody do rozłączenia, wysyłania oraz odbierania danych przez socket

Metoda „disConnectWithServer” powoduje zamknięcie połączenia oraz wszystkich otwartych zmiennych przypisanych do komunikacji między serwerem i klientem, jeśli takie połączenie istnieje.
Metoda „sendDataWithString” pobiera jeden argument, który jest wiadomością, który zamierzamy wysłać do drugiego urządzenia. W języku Java za wysyłanie wiadomości odpowiada funkcja „write”, lecz należy również w niej użyć funkcji „flush”, która ma za zadanie opróżnianie strumienia wyjściowego i wymusić zapisanie/wysłanie wszystkich zbuforowanych bajtów wyjściowych.
Metoda „receiveDataFromServer” odpowiada, za odbieranie wiadomości dostarczonych z serwera. Zwraca ona tablice bajtów, które zostały odebrane z serwera za pomocą funkcji „getInputeStream ()” oraz „read ()”.

Klasa automatic.java:

```
public class automatic extends AppCompatActivity {
Boolean runProgram = true;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_automatic);
    Button run = (Button) findViewById(R.id.automatic);
    Button stop = (Button) findViewById(R.id.stop);

    run.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            new AsyncTask<Integer, Void, Void>() {
                @Override
                protected Void doInBackground(Integer... params) {
                    runProgram = true;
                    steruj( host: "192.168.0.2",  user: "pi",  password: "raspberry",  command1: "python Desktop/Auto.py");
                    SocketConnect nc = new SocketConnect( host: "192.168.0.10",  port: 7000);
                    nc.connectWithServer();
                    String message = "Ready";
                    byte [] receive = null;
                    String str = null;
                    nc.sendDataWithString(message);
```

Rysunek 7.16. Funkcja do wykonania jazdy autonomicznej

Klasa automatic.java składa się z globalnie dostępnej zmiennej typu „Boolean”, która w dalszej części programu będzie odpowiadała, za przerwanie połączenia z serwerem dostępnym na Raspberry Pi. Następnie zaimplementowane są 2 przyciski odpowiadające za włączenie pracy autonomicznej oraz jej zakończenie. W momencie kliknięcia przycisku „run” zmiennej „runProgram” zostaje przypisana wartość „true”. Następnie uruchomiony zostaje odpowiedni skrypt na urządzeniu mobilnym, który odpowiada za wysyłanie danych z czujnika, oraz odbierania komend z aplikacji. W dalszej części kodu uruchomiona zostaje metoda do połączenia się poprzez socket z serwerem i zostaje wysłana pierwsza wiadomość „Ready”.

```
while (runProgram) {
    try {
        receive = nc.receive();
    } catch (IOException e) {
        e.printStackTrace(); }
    try {
        str = new String(receive,  charsetName: "UTF-8");
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
    List<String> czujnik = Arrays.asList(str.split( regex: ","));
    System.out.print("Rozmiar listy: " + czujnik.size() + "\n");
    Float f1 = Float.parseFloat(czujnik.get(0));
    Float f2 = Float.parseFloat(czujnik.get(1));
    Float f3 = Float.parseFloat(czujnik.get(2));
    if(f1 < 3 ) {
        nc.sendDataWithString("bacAndLeft");
    }else if(f2 < 3 ) {
        if(f1 > f3)
            nc.sendDataWithString("bacAndLeft");
        else
            nc.sendDataWithString("bacAndRight");
    }else if(f3 < 3 ) {
        nc.sendDataWithString("bacAndRight");
    }else {
        nc.sendDataWithString("Forward"); }
    String string1 = String.format("%.02f, ", f1);
    String string2 = String.format("%.02f, ", f2);
    String string3 = String.format("%.02f \n", f3);

    System.out.print(string1);
    System.out.print(string2);
    System.out.print(string3);
    System.out.print("\n"); }
nc.disConnectWithServer();
return null;
```

Rysunek 7.17. Dalsza część kodu odpowiedzialna za jazdę autonomiczną robota

Po wysłaniu pierwszej wiadomości z aplikacji do serwera, zostaje zaimplementowana pętla, która wykonuje się, dopóki, zmienna „runProgram” jest równa „true”. W pętli tej wykonują się sekwencyjnie odpowiednie komendy. Najpierw zostaje odebrana wiadomość, która zawiera dane z czujników. Jako że wiadomość ta, jest tablicą bajtów, należy ją rozszyfrować, oraz przypisać do zmiennej „String”. Wiadomości z serwera są wysyłane jako informację odległości ze wszystkich czujników, odseparowane od siebie przecinkiem. Została więc stworzona lista, która pobiera wszystkie te dane i do odpowiednich komórek przypisuje, dane z konkretnego czujnika. Następnie zmienne String są przekonwertowane na zmienną Float (czyli zmiennoprzecinkową) i przypisana do konkretnej zmiennej. Następnie w zależności od wartości, zostaje przypisana odpowiednia akcja, która zostaje wysłana do serwera. Głównym założeniem algorytmu jest przesyłanie odpowiednich danych z czujników. Jeśli z czujnika lewego lub prawego odległość wynosić będzie mniej niż 3 cm, wtedy robot cofa się, oraz skręca w odpowiednią stronę, lecz jeśli na czujniku odpowiadającego za badanie przestrzeni na wprost robota odległość będzie mniejsza niż 3 cm, wtedy program sprawdza odległości z czujnika lewego i prawego, następnie cofa się i jedzie w stronę, gdzie odległość wyniosła większą wartość. Dane nie są wyświetlone przez aktywność, lecz zostały one wypisane na tablicy służącej do debugowania aplikacji i w czasie jej uruchomienia i działania, można określić czy dane z serwera są odpowiednio wysyłane i czy podejmowane są odpowiednie kroki. Jeśli pętla zostaje przerwana, wtedy następuje również wywołanie metody odpowiedzialnej za rozłączenie pomiędzy aplikacją i serwerem.

```
11-29 16:36:27.006 30615-30703/com.example.praca.myapplication I/System.out: Rozmiar listy: 3
11-29 16:36:27.008 30615-30703/com.example.praca.myapplication I/System.out: 9,03, 6,84, 2,85
11-29 16:36:33.150 30615-30703/com.example.praca.myapplication I/System.out: Rozmiar listy: 3
11-29 16:36:33.152 30615-30703/com.example.praca.myapplication I/System.out: 5,14, 4,42, 0,45
11-29 16:36:39.294 30615-30703/com.example.praca.myapplication I/System.out: Rozmiar listy: 3
11-29 16:36:39.297 30615-30703/com.example.praca.myapplication I/System.out: 2,84, 3,78, 4,03
```

Rysunek 7.18. Przykładowe dane wysłane przez czujniki odległościowe z urządzenia mobilnego do aplikacji

```
stop.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        runProgram = false;
        new AsyncTask<Integer, Void, Void>() {
            @Override
            protected Void doInBackground(Integer... params) {
                try {
                    steruj( host: "192.168.0.2",  user: "pi",  password: "raspberry",  command1: "python Desktop/stop.py");
                } catch (Exception e) {
                    e.printStackTrace();
                }
                return null;
            }
        }.execute(1);
    }
});
```

Rysunek 7.19. Przykład funkcji odpowiedzialnej za przerwanie jazdy autonomicznej

Drugi przycisk służy do wyłączenia jazdy autonomicznej w robocie. Na samym początku wartość „runProgram” zostaje ustawiona na „false” i w tym samym momencie pętla odpowiedzialna za przesyłanie oraz odbieranie danych zostaje przerwana i następuje rozłączenie z serwerem. Następnie zostaje wywołana komenda do zatrzymania wszystkich silników.

7.5.3. SERWER NA RASPBERRY PI – PYTHON

Funkcja do jazdy autonomicznej została dodana do programu odpowiedzialnego za działanie czujników.

```
print_lock = threading.Lock()

def threaded(c):
    while True:

        data = c.recv(1024)
        if not data:
            print('Bye')

            # lock released on exit
            print_lock.release()
            break

        time.sleep(1)

        if data == 'Forward':
            print('Go Forward')
            os.system("python forward.py")
        elif data == 'bacAndLeft':
            print('BackAndLeft')
            os.system("python backAndLeft.py")
            time.sleep(5)
        elif data == 'bacAndRight':
            print('BackAndRight')
            os.system("python backAndRight.py")
            time.sleep(5)

        data1 = repr(distance1)+','
        data2 = repr(distance2)+','
        data3 = repr(distance3)

        data = data1+data1+data3

        c.send(data)

    c.close()
```

Rysunek 7.20. Funkcja odpowiedzialna za wysyłanie, odbieranie i realizację komend wysłanych z aplikacji mobilnej.

Główna funkcja odpowiada za przesyłanie, odbieranie danych oraz uruchomienia odpowiednich skryptów odpowiadających za jazdę robota. W funkcji znajduje się nieskończona pętla, w której na samym początku zostaje odebrana z aplikacji wiadomość, a jeśli nie zostanie ona dostarczona, wątek odpowiedzialny za przesyłanie wiadomości zostanie zwolniony. Po otrzymaniu wiadomości program w zależności od tego jaką wiadomość otrzymał wykonuje odpowiedni skrypt do jazdy urządzenia. Jeśli urządzenie musi wycofać się i skręcić w odpowiednią stronę, program wstrzymuje działanie na 5 s, aby dać możliwość wykonania danej operacji. Dane z czujnika, są przypisywane do odpowiednich zmiennych, wraz z niezbędnym przecinkiem, który umożliwia aplikacji zdefiniowanie, danych z czujników. Następnie dane te zostają wysłane na serwer. W momencie zakończenia działania funkcji połączenie zostaje zamknięte.

```
def Main():
    host = 'localhost'
    port = 7000
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((host, port))
    print("Socket: ", port)

    s.listen(1)
    print("Trwa wyczekiwanie")

    while True:

        c, addr = s.accept()

        print_lock.acquire()
        print('Connected to :', addr[0], ':', addr[1])

        start_new_thread(threaded, (c,))
    s.close()


if __name__ == '__main__':
    Main()
```

Rysunek 7.21. Funkcja główna na urządzeniu mobilnym służąca do utworzenia serwera, z którym łączy się aplikacja

W funkcji „Main ()” zostaje zawarte utworzenie serwera. Zostaje podany odpowiedni host oraz port, na którym połączenie zostaje utworzone. Do serwera może się połączyć jedynie jedno urządzenie, a w momencie uruchomienia pętli while, zostaje zarezerwowany wątek do komunikacji pomiędzy urządzeniami.

7.6. AKTYWNOŚĆ PIĄTA – INFORMACJE O APLIKACJI

7.6.1. REPREZENTACJA GRAFICZNA

Rysunek 7.22. Reprezentacja graficzna aktywności zawierającej podstawowe informacje o aplikacji

Ostatnia aktywność w aplikacji, ma jedynie charakter podglądowy dla jej użytkownika, aby uzyskać informację dotyczące samej aplikacji, jej podstawowych funkcji oraz autorze. Składa się ona jedynie z jednego komponentu TextView w którym zawarte są wszystkie informacje. W pliku Java nie został zawarty żaden kod.

8. PODSUMOWANIE

W obecnych czasach szybkie oraz niezawodne działanie aplikacji oraz oprogramowania pomaga nam w życiu codziennym. Aplikacje do sterowania robotami mobilnymi, urządzeniami, czy nawet wyświetlania aktualnej trasy w drodze samochodem, pozwala na odciążenie nas od wielu poważnych aspektów dnia na które bez aplikacji musielibyśmy zużyć o wiele więcej czasu. Rozwiązanie służące do sterowania manualnego oraz automatycznego przez platformę mobilną było docelowym tematem realizowanej pracy inżynierskiej. Działy poświęcone w obecnej pracy służyły do pokazania najważniejszych aspektów realizacji tych celów wraz z przesyłaniem obrazu na żywo. Na podstawie literatury ogólnie dostępnej opracowana została teoretyczna część, w której zostały opisane najważniejsze aspekty aplikacji mobilnych w Android Studio. Zostały również pokazane najważniejsze informacje dotyczące komunikacji bezpiecznej między urządzeniami oraz konfiguracje w Raspberry Pi do aktywacji danej komunikacji oraz aby móc korzystać z dołączonej kamery. Przeanalizowane zostały w dalszej części pracy konkretne funkcjonalności programu zaimplementowane w aplikacji mobilnej służące do sterowania za pomocą odpowiednich przycisków robotem, przesyłania obrazu z kamery do aplikacji oraz samej jazdy autonomicznej robota. Wszystkie funkcje zaimplementowane w aplikacji po wykonaniu testów w warunkach domowych przebiegły bezproblemowo, a interfejs dostępny dla użytkownika jest wygodny w użyciu i intuicyjny.

9. LITERATURA

1. Satya Komatineni, Dave MacLean, Sayed Hashimi, “Android 3 tworzenie aplikacji”, Wyd. 1, Wydawnictwo Helion 2012, Gliwice, ISBN 978-83-246-4184-0.
2. Marcin Płonkowski, „Android Studio Tworzenie aplikacji mobilnych”, Wydawnictwo Helion 2018, Gliwice, ISBN 978-83-283-4462-4.
3. Andrzej Stasiewicz, „Android Podstawy tworzenia aplikacji”, Wydawnictwo Helion 2013, Gliwice, ISBN 978-83-246-8841-8.
4. Allen B. Downy „Myśl w języku Python”, Wydanie 2, Gliwice, Wydawnictwo Helion 2017, ISBN 978-83-283-3003-0.
5. Cay S. Horstmann, „Java 9 – Przewodnik doświadczonego programisty”, Wydanie 2, Wydawnictwo Helion 2018, Gliwice, ISBN 978-83-283-4251-4.
6. Paul Deitel, Harvey Deitel, Alexander Wald, “Android 6 dla programistów. Techniki Tworzenia Aplikacji”, Wyd 3, Wydawnictwo Helion 2016, Gliwice, ISBN 978-83-283-2579-1.