A projekt célja, egy olyan távirányítható eszköz elkészítése volt, amely képes egy bolti

őrrobot funkcióit ellátni. Az eszköz feladata egy közepes méretű bolt bejárása, és a falakra elhelyezett QR kódokból kinyert utasítások végrehajtása.

Képes ezen kívül szükség esetén egy Androidos telefonról utasításokat fogadni (ezek többnyire a QR kódból kinyerhető utasítások) és azokat végrehajtani. A külső féltől származó utasításokat egy PC-n futó szervertől fogadja.

A kinyomtatott QR kódok a következő utasításokat tartalmazzák:

1. **FORWARD:TRUE :** Ilyenkor PWM vezérelve elindul az autó, annak érdekében, hogy a további kódokat könnyebben be tudja olvasni csak 30%-os kitöltöttségű PWM-et van beállítva, de ez módosítható
2. **FORWARD:FALSE :** Ilyenkor az autó abbahagyja az előre haladást. A motorban található fogaskerekeknél fellépő súrlódás miatt azonnal megáll, nem szükséges fékezni de egy 10%-os kitöltöttségű tolatással ez is elérhető.
3. **BACKWARD:TRUE :** Ilyenkor PWM vezérelve elindul az autó hátra. A motorban található fogaskerekek kopottságából adódóan itt 50%-os PWM-et kellett alkalmazni, de ez is módosítható.
4. **BACKWARD:FALSE** :Ilyenkor az autó abbahagyja a tolatást.
5. **LEFT:TRUE** : Ilyenkor elforgatja a kerekeket PWM vezérelve balra ügyelve arra, hogy ne terhelje a végpont elérése után a kormányzásért felelős motort. Amennyiben a kormány a jobboldali maximális végállásban van akkor egyenesbe forgatja a kereket.

**LEFT:FALSE :** Ez egy elkészített, de sosem használt funkció. Abbahagyja a kerekek balra forgatását.

1. **RIGHT:TRUE :**  Ilyenkor elforgatja a kerekeket PWM vezérelve jobbra ügyelve arra, hogy ne terhelje a végpont elérése után a kormányzásért felelős motort. Amennyiben a kormány a baloldali maximális végállásban van akkor egyenesbe forgatja a kereket.
2. **RIGHT: FALSE** : Ez egy elkészített, de sosem használt funkció. Abbahagyja a kerekek jobbra forgatását.
3. **PICTURE\_SEND:** Fejlesztés alatt. Készít egy képet az erre a célra felszerelt kamerával és elküldi a kliensnek.

A robot vezeték nélküli internet segítségével képes kapcsolódni egy szerverre, de csak helyi hálózat (LAN) esetén. A szerver fejlesztése Windows operációs rendszer alatt történt NetBeans, Maven felhasználásával. A további fejlesztési fázisokban Linux operációs rendszer alatt volt használva, nincs tapasztalat működést megakadályozó hibával kapcsolatban.

A szerver célja, hogy igény esetén több robot és telefon-kliens is képes egymással kommunikálni, ezen kívül a képküldés is itt valósul meg, így a bejövő adatok szabadon módosíthatóak például.: a készített képeket a szerveren is le lehet tárolni. A beérkező képet továbbítja a célként kiválasztott kliensre.

**Programozási nyelvek**:

Java**:**

A Java egy objektumorientált programozási nyelv, amelyet a Sun Microsystems a '90-es években fejlesztett egészen 2009-ig amikor az Oracle tulajdonába került a cég. A szerveren és a kliensen - mind az Androidos mind a PC kliens beleértve - Java nyelven készült el a kód. A nyelv egyik nagy előnye a platformfüggetlenség. A megírt kód hardvertől és operációs rendszertől függetlenül fut a Java Virtual Machine-nek (JVM) köszönhetően, mert a fordító bájtkódra fordít. A bájtkód közvetlenül futtatható a processzoron. A nyelv a szintaxisát a C valamint a C++ nyelvtől örökölte. Ezt kihasználva lehet a kódot apróbb módosításokkal átültetni x86 alapú PC-ről az ARM processzoros Android operációs rendszerrel rendelkező telefonra, és a Raspberry Pi 4-re.

A projekt jellegéből adódóan szükséges volt az objektumorientáltság használata. Az elképzelés szerint a szerver a különböző folyamatokat ( adatok fogadása a klienstől, adatok továbbítása, szálak leállítása és indítása, kliensek kezelésének szétválasztása ) különböző objektumokként kezeli amelyek külön szálakon futnak és állnak le ha szükséges.

A JVM-et a Linux rendszerekre ( Ubuntu, Raspberry Pi OS ) nem, de a Windowsra telepíteni kell, mert nem tartozéka az alap szoftvercsomagnak. A Java Standard Edition (Java SE) nem része egyik rendszernek sem, így a szoftver fejlesztéséhez ezeket is be kellett szerezni. (#Wikipédia használva#)

Python**:**

A Python egy általános célú magas szintű programozási nyelv. A tervezési filozófia az olvashatóságot és az egyszerű alkalmazást tartja fontosnak, a futási sebességgel szemben. Funkcionális és Objektumorientált paradigmákat is támogatja. A típusokat és a memóriát dinamikusan kezeli. Interpreteres nyelv, így nincs a tárgy és a forráskód különválasztva. Python értelmező segítségével azonnal futtatható. Az értelmezőt számos operációs rendszerre elkészítették így széles körben alkalmazható.

A Raspberry-n futó klienst újra kellett írni, mert nem volt alkalmas a kamera integrálására OpenCV-vel. Az új verzió Python nyelven készült el. A program szálainak funkciói majdnem teljesen megegyeznek a Java prototípuséval, így sok felmerülő kérdést, problémát kész megoldásokkal lehetett elhárítani.

A Pyhton kiválóan alkalmasnak bizonyult a Java-nál felmerülő problémák orvosolására. Meg lehet valósítani az objektumorientáltságot, szálkezelést, socket programozást és az OpenCV segítségével integrálható a kamera modul is.

A Raspberry Pi OS rendszer alapértelmezetten támogatja a Python3-at, de szükséges volt az OpenCV telepítése, ami parancssorral könnyen megoldható. (A telepítésnél vigyázni kell, mert az OpenCV legújabb verziója nem telepíthető Raspberry-re így a projekt 3.4.16-ot használ az aktuális 4.5.4 helyett. ) A Python meglehetősen pazarlóan bánik az erőforrásokkal, így az első próbálkozások a futtatással nagyon magas CPU használat volt megfigyelhető. Ezt a problémát a megfelelő kamerák kiválasztásával ki lehetett javítani. (#Wikipédia használva# nem baj, csak legyen hivatkozás)

OpenCV**:**

Az OpenCV egy C++ nyelven írt ingyenes szoftveres képfeldolgozásért felelős függvénykönyvtár. A C++ alapok ellenére elérhető Python, Java és Matlab nyelven is. Ezen interfészek API-ja az online dokumentációkban érhető el.

A projekten belül a Raspberry-n futó Python kliens használja a QR kódok tartalmának azonosítására. Ehhez, a cv2.QRDetector-t kellett alkalmazni. Könnyen integrálható python3-ban, képes kezelni az USB-n és a Raspberry alaplapján található kamera porton keresztül kapcsolt webkamerákat is. A tesztek alapján a Raspberry egy 600 x 800-as felbontású kamera képét tudja a legkönnyebben feldolgozni. Ez az aranyközépút, mert a nagyobb felbontással könnyebben megtalálja a szoftver az objektumokat, de a több adatot lassabban dolgozza fel, míg a kisebb felbontásnál gyorsabb a bejövő képek ellenőrzése de a kevesebb részlet miatt nehezebben azonosítja az objektumokat.

A bejövő képeket a kívánt formátumban le lehet menteni a háttértárra (vagy socketen keresztül továbbítani). Mivel a képek mentése .png formátumban történik, a szerver és a telefon kliens oldalon nem szükséges az OpenCV használata. (#Wikipédia használva# nem baj, csak legyen hivatkozás)

**Socket programozás:**

Socket: A számítógép-hálózatokban a socket egy Internet Protocol-alapú hálózati folyam eleme. Valamilyen számítógépes hálózatban például az interneten kétirányú adatkommunikáció végpontja.

A kifejezést a TCP/IP protokollkészlet (általában az operációs rendszer által biztosított) API-jának megnevezésére is használják. Az internet socketek hozzárendelik az adatcsomagokat a megfelelő alkalmazásokhoz és szálakhoz, a megfelelő port és ip kombinációjának segítségével.

Socket elemei:

1. A socket címe ami egy IP cím ami egy eszközhöz csatolja
2. A socket portja ami egy alkalmazáshoz kapcsolja az adatokat. Ezzel a két adattal egyszerűen azonosítható az interneten az alkalmazások.

Socketek típusai:

1. Raw socket: routerekben és egyéb hálózati eszközökben találhatóak. Itt a csomagok fejléce nincs levágva, az alkalmazások számára hozzáférhető kimarad a szállítási réteg
2. Stream sokcet: ezek a TCP protokoll szerint működnek, kapcsolat-orientáltak. Az adatcsomagok sorrendhelyesen továbbítódnak.
3. Datagram socket: kapcsolatmentes adattovábbításra valóak, az User Datagram Protocol-t (UDP) alkalmazzák. Az adatcsomagok nem sorrendhelyesen továbbítódnak.

Szerver:

Egy szerveroldali socket kommunikációt valósít meg. Fogadja a kliensek kéréseit és továbbítja a megfelelő cél kliensnek.

Kezdetben a "listening" állapotban várakozik, ilyenkor a netstat által jelzett távoli cím 0.0.0.0 a port pedig 0. Megszólítás után veszi fel a kliens adatait, majd egy új szálon "listening" állapottal indít egy új szerver socketet.

A szerver tehát egyidejűleg több különálló socketet is tud kezelni. Az operációs rendszer ezeket különálló socketekként kezeli. TCP kapcsolat esetén azonos IP-hez és porthoz több socket is tartozhat.

Kliens:

Egy kliensoldali socket kommunikációt valósít meg. Fogadja a szerverektől kapott üzeneteket és a bejövő adatok alapján végrehajtja a kívánt utasításokat majd válaszol a szervernek. Ismeri a szerver IP-címét és a megfelelő portot és erre küldi az adatcsomagokat. A kapcsolatot a szerver megszólításával kezdi, majd a csatlakozás után a TCP protokoll szerint kommunikál a szerverrel, azon keresztül a másik klienssel.

A kliens 1 socket szálat kezel, ezen keresztül történik minden adatcsere. Nincs is szükség többre, hiszen egy klienshez egy szerver tartozik. UDP kapcsolat esetén nincs állandó kapcsolat, az adatok nem sorrendhelyesen továbbítódnak. A TCP kapcsolat esetén egyszer létrejön a kapcsolat majd ezen a csatornán keresztül történik az adatcsere.

A socket kommunikáció szempontjából nincs jelentősége, hogy Python vagy Java nyelven íródott a programkód

**Eszköz céljának bemutatása:**

A robotnak képesnek kell lennie egy közepes méretű boltban egy előre meghatározott pályán haladnia és szükség esetén fényképeket készítenie a beépített kamerával. Egy gyengébb felbontású kamera segítségével haladás közben dolgozza fel az információkat, amelyeket QR kódokon keresztül kap meg. A másik nagyobb felbontású kamerával fényképeket készít és menti vagy továbbítja a szerver és a másik kliens felé.

A sebesség a PWM érték módosításával befolyásolható. Az alapértelmezett PWM kitöltés 30%. Az egyharmados kitöltöttség az áramforrás élettartamára is kedvezően hat. A PWM mértékét befolyásolja a talaj anyaga is, szőnyegen sokkal nagyobb teljesítmény szükséges mint parkettán. A fogaskerekek kopottsága miatt, az előre és a hátra haladáshoz szükséges kitöltési tényező nem egyenlő. ( Hátra könnyebben halad az autó. )

Az irányváltáshoz egy villanymotort használ, amit szintén PWM elven lehet vezérelni. Itt magasabb kitöltési tényezőt kell használni, mert a kerekek forgatása közben a súrlódás nagy. A kerekek forgatása is nagyban függ a talajtól. A gumiabroncs a legnagyobb súrlódást a lakkozott parkettán fejti ki, míg a legkisebbet a csempén. Ha a kerekek elérték a maximális kitérést az egyenestől, a szoftver nem enged további kerékszögváltozást az adott irányba. A teljesen kifordított kerék előbb egyenes állapotba áll vissza ha utasítást kap ( például.: max bal + jobb = egyenes ) és még egy utasítás szükséges a másik szélsőértékre való beálláshoz.

A kamera képe alapján képes a QR kód távolságának meghatározására, ezt a bejövő kép és a QR kód méretének arányából számítja, 2cm-es pontossággal. A minimális QR kód beolvasási távolság a kamerától függ. A maximális QR beolvasási távolság nagyjából 230 cm ezt a kamera sajátosságai befolyásolják. Kamera legnagyobb látótávját szoftveresen is lehet állítani. A fényviszonyok, és a beolvasandó objektum mérete és minősége is hatással van a beolvasás sikerességére, ezért erre a felhasználási területen figyelni kell. A felismerési ráta maximalizálására a legjobb körülmények a hideg fényű fényforrások és a stabilan elhelyezett QR kóddal ellátott táblák, valamint a stabil, folytonos rázós felületektől mentes talaj. Amennyiben a talaj nem megfelelő a vertikális irányú rázkódás sokszor használhatatlanná teszi az adott képkockát.

A tápegység a környezettől és a végrehajtandó utasításoktól függően tud 5V egyenfeszültséget biztosítani a Raspberry-nek. Ideális esetben, amit a sima talaj és a nagy, jól felismerhető táblák, nagy egyenesek kevés kanyar, jelentenek akár másfél órás üzemidő is elérhető. Használható akkumulátor is, mert az alapot képező távirányítós autó elemmel is és akkumulátorral is működtethető.

A korai tesztekben a Raspberry tápját egy hordozható 5V 2A 10 000mAh teljesítményű akkumulátor látta el. Ez nem bizonyult hatékony konstrukciónak, mert a nagy súly miatt nem tudott mozogni megfelelően.

A fénykép elkészítése két módszerrel történhet:

1. Megállás után elkészül a fotó majd tovább halad. Ezzel a módszerrel minimalizálható az elmosódás, de időigényesebb a feladat végrehajtása. Célszerű rosszabb látási viszonyok esetén használni. Álló és mozgó objektumokról szerzett információk rögzítésére is egyaránt alkalmas.
2. Haladás közben is készülhet a fénykép. Ezzel a módszerrel viszont van esély elmosódásra ami az adott képkocka használhatatlanságát is okozhatja. A végrehajtás kevésbé időigényes, de a minőség is gyengébb. A tesztek alapján többnyire álló objektumok fényképezésére alkalmas.

Amennyiben haladás közben készül a fénykép, figyelni kell arra, hogy nem szabad kanyarodás közben fotózni. Ilyenkor 2 irányú lehet az elmosódás. Mert a fordulás horizontálisan a rázkódás pedig vertikálisan okoz elcsúszást.

Az eredeti koncepció szerint a kódolvasó kamera a földet pásztázta volna, de az a magasság ami az autón maximálisan elérhető nem volt elegendő a megfelelő fókusztávolság beállítására, így az autó 2 kamerája előre néz.

**Felmerült problémák és megoldásaik:**

A fejlesztés közben felmerül első probléma a socket kapcsolat portjaival volt. Kis kutatás után kiderült, hogy az eredetileg használt portot (1000) egy másik alkalmazás használja így a szerver a 10000-res porton fut. Ezzel a beállítással a portok beállításai megfelelőnek bizonyultak. Egy VPN felhasználásával LAN függetlenné tehető a rendszer.

A kliensek azonosítása, a kliens "megszólítása" után történik. Miután létrejött a kapcsolat a kliens elküldi a felhasználónevét. A szerver indít egy új szálat, ami a "listening" állapotba kerül. A kliens minden üzenetbe belekódolja a cél kliens felhasználónevét, ezek alapján tudja a szerver ki a cél. Majd törli a tárolt üzenetet, ha ez elmarad akkor addig küldi ki az utolsó kapott üzenetet amíg mást nem kap.

A kommunikáció során előfordult olyan eset, hogy lecsatlakozás után a szerver oldalon a socket és az őt életben tartó szál nem állt le, ezzel értékes erőforrásokat pazarolva. A megoldás az volt, hogy időközönként a szerver küld egy ellenőrző üzenetet amire a kliens válaszol. Ha ezt nem kapja meg akkor leállítja az adott szálat. Újracsatlakozás után az adatok hiba nélkül továbbítódhatnak. A mérések alapján 10 klienst lehet egy átlagos teljesítményű számítógépen egyszerre használni. A hiba javítása után 10 klienssel még nem volt probléma.

A Raspberry-n a Java nyelven nem működött az OpenCV a probléma oka a Raspbian operációs rendszer c++ fordítója volt. Nem felelt meg a verziószám az előírt minimálisnak. 11-es vagy magasabb verziójú fordító szükséges hozzá. A megoldás a Python nyelv használata volt Java helyett amivel a telepített OpenCV már probléma mentesen futott.

A Python változat nagyjából azonos módon működik mint a Java. Az osztályokat és a függvényeket át lehetett ültetni. A funkcióik azonosak, a változó és az osztálynevek hasonlóak. Először a socket részek készültek el majd a többszálas futtatás végül a kamera integrálása a rendszerbe végül a QR kódfelismerés.

A Raspberry-re a Python legutolsó verziója (4.5.4) ismeretlen okból nem települ így a 3.4.16-os verzióval készült el a szoftver. Ez a működésre nincs hatással. A Python csomagtelepítője az utolsó lépésnél megakad és hosszas idő után hibaüzenettel leáll a legfrissebb verzió esetén.

A rendszer két kamerát tartalmaz. Az eredeti tervek szerint az egyik a földet nézte volna és ott keres QR kódot. Ha fölé ér akkor végrehajtja az utasítást. A probléma itt az volt, hogy az autónak minden fordulás után pont a kód fölé kell érnie és a kamerának a fókusztávolsága nagyobb volt mint az elérhető maximális magasság. A következő lehetőség az lett volna, hogy a kamera az autó tetején van és előre néz. A földre elhelyezett kódot kellett volna felismerne. A földön lévő tábla és a 45 fokban elforgatott kamera együttesen okozta, hogy a bejövőkép vertikálisan torzul. Az így módosult objektum nem olvasható be. A megoldás a táblákra és falakra elhelyezett papírlapok és rajtuk a nyomtatott ábrák. A másik kamera jobb minőségű kép rögzítésére alkalmas. Ezzel az eszközzel lehet fényképeket készíteni. Azért nem ez az alapértelmezett kamera, mert a nagyobb felbontás miatt nagyon megnöveli az erőforrásigényt, ami az ARM alapú Raspberry-nél erősen korlátos.

A Python egy interpreteres nyelv, ellentétben a fordított nyelvekkel mint például a C. A fordítóval rendelkező nyelveket futtatás előtt gép kóddá kell alakítani és csak ezt követően lehet átadni a processzornak. Az interpreteres nyelvek futási időben fordulnak át gépkóddá. Ez is hozzájárul a nagyobb rendszerigényhez.

A megfelelő mozgási sebesség eléréséhez impulzusszélesség-modulációt (PWM) kell alkalmazni, mert a teljes sebességgel való haladás negatívan hat az elérhető üzemidőre és a kamera képét is nehezebb értelmezni. A motorok teljesítményszintjének állításával könnyebben megvalósítható a fokozatos kanyarodás is. Vigyázni kell, mert minimum 15%-os kitöltési tényező kell az előrehaladáshoz és minimum 30% a kerekek forgatásához. Ennél kisebb értékek esetén a motorok nem képesek ellátni a feladatukat. ez korábbra is mehet

**Raspberry kliens bemutatása:**

Az egylapkás rendszer, vagy System on Chip ( SoC / SOC ) egy olyan integrált áramkör, amely egy számítógép, vagy más elektronikai rendszer összes komponensét egyetlen lapkán tartalmazza.

Egyes nagyobb teljesítményű változataik Linux vagy Windows rendszereket is futtathatnak, ha kompatibilisek a szükséges processzorcsaláddal, és a minimálisan előírt hardveres elvárásokat képesek teljesíteni.

Az egylapkás rendszer célja, az integráció fokának növelése a gyártási költségek csökkentése és a kisebb rendszerek létrehozására való törekvés.

A Raspberry Pi egy bankkártya méretű BCM2835 alapú egylapkás rendszer. Különböző Linux disztribúciók futtatására képes.

A hivatalosan ajánlott operációs rendszer a laphoz a Raspberry Pi OS ( korábban Raspbian ), ami a Debian Linux kifejezetten Raspberry Pi-re optimalizált változata.

Minden Raspberry Pi modellben integrálva megtalálható:

* CPU
* RAM
* GPU
* USB port
* MIPI Kamera interfész
* HDMI port
* 3,5 mm jack port
* memória kártya foglalat
* GPIO lábak
* 5V tápcsatlakozás

Az USB, HDMI GPIO portok mennyisége és minősége a különböző modellek között eltérő lehet. A CPU sebessége, a RAM mennyisége is változhat a modelltől függően. Az egységek fizikai mérete közel azonos, a Pi Zero-t leszámítva ( ez körülbelül feleakkora ).

Néhány Raspberry Pi eszköz jellemzője:

1. Pi Zero: alacsony teljesítményű és árú eszköz, 1Ghz-es 1 magos processzorral és 512MiB RAM-mal, HDMI, USB és micro-USB porttal. Valamit egy 40 pines foglalattal, amely tartalmaz 26 darab GPIO, 2darab 3.3V-os, 2darab 5V-os és 8darab test lábat.
2. Pi 1 - 3: Normál teljesítményű, 700Mhz - 1.2Ghz processzorral, 512MB - 1024MB RAM-mal, HDMI, USB, micro-USB porttal szerelt. Tartalmaz egy 40 pines GPIO foglalatot. Wifi és Bluetooth kapcsolatra is képes.
3. Pi 4: Az elődeinél nagyobb teljesítményű. 4 magos Cortex-A72 64bites 1.5Ghz-es processzorral, 2 - 8 GB LPDDR4-3200 SDRAM-mal szerelt rendszer. 2.4 GHz és 5.0GHz IEE 802.11ac wifi, Bluetooth 5.0, BLE Gigabit Ethernet biztosítja a kommunikációt a többi eszközzel. 2 db USB 3.0, 2 db USB 2.0, 2 db micro-HDMI, MIPI DSI kijelző, MIPI CSI kamera port található rajta. A robotot egy ilyen integrált lapja vezérli.

Meg lehet különböztetni A / A+ / B / B+ modelleket. Ezek többnyire a portok számában, vagy azok teljesítményében különböznek.

**GPIO ( General-purpose input/output ):**

Általános célú bemeneti / kimeneti csatlakozó. Egy integrált, vagy elektromos áramkörön biztosít digitális csatlakozást a lábakon keresztül. Használható bejövő jelek fogadására és kimenő jelek vezérlésére is. Képes a felhasználó futási időben vezérelni.

A GPIO lábaknak ( pin ) nincs előre meghatározva a felhasználás célja. A felhasználási célt az áramkör felhasználója határozza meg.

Több alkalmazási mód is lehetséges. Csak a GPIO láb elektromos időzítése, a szoftver sebessége befolyásolja a sebességét. Szabványos logikai jelszinteket használ, nem képes nagy teljesítmény átadására, tehát csak a vezérlő jeleket lehet vezérelni.

Néhány alkalmazási terület:

• LED állapotjelzők vezérlése

• más áramkörök működésének engedélyezése és letiltása

• fedélzeti kapcsolók leolvasása

• konfigurációs söntök leolvasása

A GPIO-k alapvetően digitálisak, de gyakran használják őket lineáris folyamatok vezérlésére. Ilyen eset például a projekten a mozgást és fordulást végző motorok vezérlése. Ezt PWM-en keresztül lehet megtenni. A kimenő jel munkaciklusa határozza meg a folyamatvezérlő jel effektív nagyságát.

**Raspberry kliens Java prototípus:**

A Raspberry-n futó kliens első változata, egy java nyelven írt program. A java kód könnyedén mozgatható a különböző operációs rendszerek között (Windows, Ubuntu, Raspberry OS, Android), így sok időt lehetett nyerni a fejlesztésével. Java swing elemekkel készült hozzá egy egyszer grafikus felület is, amely csak a felhasználást könnyíti meg de az összes funkció elérhető a konzolablakból is.

Indulás után bekéri a szerver IP-címét, a saját és a cél kliens felhasználónevét. Megkísérel csatlakozni a szerverre, ha ez sikeres elküldi a saját felhasználónevét, hogy a későbbiekben azonosítani lehessen. Ehhez socket kapcsolatot használ.

A kapcsolódáshoz szükséges adatokat alapértelmezetten a grafikus felületről kéri be, a nem megfelelő vagy hiányos kitöltés esetén figyelmezteti a felhasználót.

Tesztelési céllal lehetőség van szöveges üzenetek küldésére és fogadására. A felhasználó a küldeni kívánt üzenetet egy erre a célra elhelyezett szövegdobozba írhatja be amit a küldés gombbal tud a szerverre eljuttatni.

A kliens képes a fogadott üzenetek elemzésére. Ha a bejövő üzenet végrehajtható utasítást tartalmaz akkor végrehajtja azt. Ilyenkor az előre beállított GPIO lábakat vezérli. Mivel ez a kód csak a fejlesztés első szakaszában volt használva, ezért nincs benne PWM, de a végleges változatban már szerepel.

A GPIO lábak vezérléshez a következő csomagokat kell használni:

com.pi4j.io.gpio.GpioController;

com.pi4j.io.gpio.GpioFactory;

com.pi4j.io.gpio.GpioPinDigitalOutput;

com.pi4j.io.gpio.PinState;

com.pi4j.io.gpio.RaspiPin;

Ezek a csomagokat az internetről kell letölteni és feltelepíteni a Linux rendszernek megfelelően. Java Runtime, WiringPi Native Library szükséges hozzá.

A kódba való importálás után a dokumentációnak megfelelően használható. Be kell állítani a vezérelni kívánt lábakat, majd "Pedla.toggle()" (bekapcsol) vagy "Pelda.low()" (kikapcsol) utasítással lehet az állapotukat vezérelni.

A Raspberry-re rögzített kamera képének feldolgozását, OpenCV segítségével terveztem megvalósítani, de ennek Java környezetbe való integrálása akadályokba ütközött. Gondok adódtak a telepítés során, majd a kamerát nem észlelte a csomag. Rövid kutatás után arra a döntésre jutottam, hogy az OpenCV Python nyelvvel való könnyű integrációja miatt elkészítem a kliens Python változatát.

**Raspberry kliens Python:**

A második változat Python nyelven íródott. A Raspberry OS alapértelmezetten biztosít a Python nyelvű fejlesztéshez minden szükséges eszközt (Thonny, Python fordító). Ennek a változatnak nagyobb erőforrásigénye van, így az újraírás során erre figyelni kellett.

A Python könnyű érthetősége és egyszerű felépítése miatt nagyon hamar elsajátítható. Így a kliens viszonylag gyorsan elkészült. Ez annak is köszönhető, hogy a Java változat szerkezeti elemeiből sokat megtartott, így a logikát már csak át kellett ültetni egy másik programozási nyelvre. Ennél a változatnál már cél volt a kód átláthatóbbá tétele is, így a logikailag összekapcsolódó elemek csoportosítva lettek a nekik megfelelő osztályokba és az azoknak megfelelő forrásfájlokba.

A változat már tartalmazza az OpenCV csomagokat. Képes a kamera képét feldolgozni, igény szerint egy ablakban megjeleníteni. Az egyes képkockákon megtalálja a QR-kódokat és szöveges formátummá alakítja a tartalmukat.

Az OpenCV integrálása a Python nyelvbe az interneten található dokumentáció szerint történt:

1. Telepíteni kell az OpenCV csomagot a "pip" parancs segítségével
2. Ha szükséges az OpenCV korábbi változatait ennél a lépésnél lehet kiválasztani
3. (A „pip” a Python nyelv csomagkezelő szoftvere a segítségével lehet új csomagokat beszerezni)
4. A telepítés végeztével az OpenCV importálhatóvá válik a Python számára "import cv2"
5. Importálás után egy objektumot beállítunk a kamera stream-re (0 a beépített kamera 1 az USB-s külső eszköz)
6. Beolvassuk az adatokat a kamerától
7. A legegyszerűbb módszer a tesztelésre ha megjelenítjük a képet

A Python kliens is több szálat használ, amelyek kommunikálnak egymással. Az egyes szálak funkciója megegyezik a Java változatban megtalálható változataikkal. Ennek a feladatelosztásnak az egyik nagy előnye, hogy így ha az egyik folyamat valamilyen nem várt esemény miatt leáll, akkor nem okozza az egész program leállását. Egy esetleges új folyamattal akár figyelni is lehetne a szálakat és ha hibát észlel akkor a problémás folyamatokat újraindítja. Amennyiben a probléma ismételten jelentkezik akkor tájékoztatni lehetne a felhasználót vagy a fejlesztőt róla. Ez megkönnyíti a hibajavítás folyamatát.

A jelenlegi változat is tartalmaz egy kezdetleges hibakezelést, de csak az olyan eseményeket tudja kezelni amiket ismer. Egy váratlan kamera leállást például nem képes felismerni (például az eszköz haladás közben beleakad valamilyen tereptárgyba és kihúzódik a kábel).

Képes a szerverről való lecsatlakozást észlelni és erről a felhasználót tájékoztatni. Amennyiben a kamera a futtatás elejétől kezdve nem elérhető, vagy nincs megfelelően beállítva erről is értesíti a felhasználót.

**Szoftver felépítése:**

1. Main\_class: Az összes további osztály és szál szülője. Feladata a szükséges konfigurációs lépések majd a funkciók végrehajtásáért felelős szálak elindítása.
   1. Tájékoztatja a felhasználót az alap információkról, segít a helyes konfigurálásban.
   2. Ellenőrzi az adatok helyességét, de akár úgy is beállítható, hogy mindent csináljon meg automatikusan. Mindezt parancssorból szöveges formában végzi.
   3. Csatlakozik a szerverre (erről tájékoztatja is a felhasználót).
   4. Elindítja a küldőszálat
   5. Elindítja a fogadó szálat
   6. Elindítja a kamerakezelő szálat
   7. A szálaknak átadja a közös adatokat tároló objektumot, amely minden olyan adatot tárol amit egynél több szálnak kell használnia.
2. Static\_variables: Egy olyan objektum osztálya, amely minden szálak közötti kommunikációhoz szükséges adatot tartalmaz. A benne található adattagokat az összes futó folyamat meg tudja figyelni, képes módosítani. A tárolt információk:
   1. mozgással kapcsolatos adatok
   2. a hálózati kommunikációhoz szükséges adatok (ip, port, felhasználónév, célkliens felhasználóneve)
   3. első kerekek kitérésnek értéke
   4. mozgás és fordulás PWM kitöltési tényezője
   5. GPIO vezérléshez szükséges lábak
   6. bejövő kimenő adatok
3. Reciever\_class: Feladata a szervertől érkező utasítások fogadása. Folyamatosan ellenőrzi , hogy érkezik-e valamilyen adat. Ha van bejövő üzenet akkor azt dekódolja UTF-8-as kódolásnak megfelelően, majd a mindenki által elérhető objektum megfelelő elemét beállítja a bejövő üzenetnek megfelelően. Szabályos leálláskor értesíti a felhasználót egy "Szerveroldali fogadószál leáll!" üzenettel.
4. Sender\_class: Feladata az adatok továbbítása a szerver felé a socketen keresztül. Indulás esetén értesíti a felhasználót a "Küldő és bemenet kezelő szál elindul!" üzenettel. Szabályos leállás esetén értesíti a felhasználót a leállásról a "Küldő és bemenet kezelő szál leáll!" üzenettel. Folyamatosan figyeli, hogy, van-e valamilyen küldendő üzenet a stat\_var objektumban. Küldéskor a célkliens felhasználónevét beilleszti az üzenet elejére.
5. Camera\_handler\_class: Feladata a kamerából érkező adatok feldolgozása. Indulásakor a „Kamera kezelő szál elindult!” üzenettel tájékoztatja a felhasználót.

* A felhasználó által beállított kamerából érkező adatokat beolvassa (OpenCV felhasználásával)
* A felhasználói beállítástól függően megjeleníti a "látottakat" egy új ablakban.
* QR kódot keres az adott képkockában.
* Ha talál arról tájékoztatja a felhasználót.
* Beírja az üzenetet a stat\_var qr\_data\_string mezőjébe és a data\_found\_bool mező értékét igazra állítja. Utóbbi lépésnek a kimenetkezelő szálnál lesz jelentősége.
* Amennyiben nem talál adatot akkor a qr\_data\_string mező értékét üresre, a qr\_data\_found\_bool értékét pedig hamisra állítja.
* Ha a felhasználó kéri a kamera képének megjelenítését, akkor a képen található QR kódot bekeretezi.
* A QR kód méretét és a bejövő kép méretét összevetve képes kiszámolni a kód távolságát. Ehhez a lépéshez szükség lesz még egy szorzótényezőre is, amely a kamera látószögétől függ. Ezt a tényezőt a legkönnyebben úgy lehet kiszámolni, hogy 100cm-re helyezzük a QR kódot a kamerától. A szoftver kiszámolja a QR kód és a képkocka arányát majd ezt az értéket megszorozzuk annyival, hogy 100-at kapjunk. Ezzel a módszerrel nagyjából 2,5m-ig pontos számításokat kapunk. Utána már nem elég a webkamera felbontása a stabil QR kód értelmezéshez.

Sikeres leálláskor a „Kamera kezelő szál leáll!” üzenettel tájékoztatja a felhasználót.

1. Client\_output\_thread\_class: Feladata a kliens eszköz kimeneteinek és az azokhoz szükséges beállításoknak a kezelése. Mivel a robot mozgásra képes ezért ez is egy kimenet forma. Meghívja a GPIO lábakat beállító függvényt. (Ez a függvény egy önellenőrzést is végrehajt a program indulásakor. ) Ha a bejövő üzenetek közül, vagy a kamera kezelő folyamat által talált QR kódban végrehajtandó üzenet van akkor az azoknak megfelelő függvényeket meghívja. Ilyenkor a kerékforgatás vagy a mozgás idejére indít egy új szálat ami végrehajtja a kívánt utasítást. Majd azt a változót ami az adott utasítást tartalmazza "False"-ra állítja. Ha a felhasználó szeretne a kerekek irányán manuálisan állítani azt is megteheti ilyenkor a manual\_wheel\_setter művelet hívódik meg. Itt a felhasználó a kerekek irányát finomhangolhatja, hogy a null érték pontosan előre nézzen.
2. GPIO\_output\_handler\_thread\_class: Feladata a GPIO lábak beállítása a megfelelő portokon valamint az induláskor történő önellenőrző folyamat vezérlése. Ilyenkor a robot előre - hátra mozog, majd a kerekeket jobbra és balra forgatja majd vissza egyenesbe. Ezekről a lépésekről tájékoztatja is a felhasználót. A felhasználónak lehetősége van a fentebb említett manuális kerékbeállításhoz. Ilyenkor a 'j' és a 'b' billentyűk segítségével 0,01s időre tudja forgatni a kerekeket jobbra és balra. Ha a kerekek iránya megfelelő akkor a 'q' billentyű megnyomásával tud kilépni a menüből.
3. Move\_thread\_class: Feladata a GPIO lábak vezérlése az előre vagy hátra haladás érdekében. Figyeli a stat\_var objektum forward\_bool és backward\_bool változókat. Megvalósítja a motor PWM elvű vezérlését. Az előre beállított kitöltési értéket tartja, a GPIO láb időzített be és kikapcsolásával. Miután végzett a vezérléssel leáll, ezzel is kímélve az erőforrásokat. Leállás előtt az általa vezérelt lábakat alaphelyzetbe, azaz kikapcsolt állapotba teszi. Ez egy egyszerű védelmi intézkedés amivel el lehet kerülni a telep gyors merülését és a motor leégését.
4. Turn\_thread\_class: Feladata a GPIO lábak vezérlése a jobbra vagy balra fordulás érdekében. Figyeli a stat\_var objektum right\_bool és left\_bool változókat. Megvalósítja a motor PWM elvű vezérlését. Az előre beállított kitöltési értéket tartja, a GPIO láb időzített be és kikapcsolásával. A fordulást több szakaszban oldja meg, de a vezérlés megoldható akár a QR kódba írt olyan utasítással is ami a fordulás mértékét befolyásolja, de nem ez az alapértelmezett. Az alapértékből a fordulás a szélső értékig történik. Onnan 2 ellenkező irányú fordítási utasítás kell, hogy a másik szélső értékre elérjen. Például ha jobbszélső állásban van: balra, balra utasítást kell kapnia mert először az alapállapotba kerül majd onnan fordul a baloldali szélsőértékhez. Miután végzett a vezérlés leáll, ezzel is kímélve az erőforrásokat. Leállás előtt az általa vezérelt lábakat alaphelyzetbe, azaz kikapcsolt állapotba teszi. Ez egy egyszerű védelmi intézkedés amivel el lehet kerülni a telep gyors merülését és a motor leégését.

**Autó részegységei**

A robot 3 fő komponensből áll. A vezérlést végző Raspberry-ből, a távirányítós autóból és a benne található motorokból, valamint a kettőt összekötő vezérlőelektronikából.

**L298 motorvezérlő:** Egy integrált monolitikus vezérlő. Magas feszültségű, magas áramú vezérlő ami a standard TTL logikai szintekhez készült. Feladata léptetőmotorok, mágnesszelepek, relék vezérlése. Két bemeneti foglalta van amelyeken keresztül lehet engedélyezni, vagy tiltani a rájuk kötött eszközöket. A bemeneteket a különböző jelszintekkel lehet vezérelni, amit a GPIO lábakon keresztül végez a Raspberry.

(# L298 block diagram #)

Az L298 integráltan tartalmaz két teljesítménykimenetet. A teljesítmény kimenet vezérelhető PWM elven vagy folyamatosan, ez csak a bemenetek állapotától függ. Minden híd kapcsolatot a négy bemenet állapota határoz meg, melyek az In1 ; In2 ; EnA és In3 ; In4 ; EnB. Az In bemenet vezérli a híd állapotát akkor, ha az En kapcsoló is engedélyezve van ( magas feszültség érték van rajta ). Alacsony feszültségi érték, azaz tiltás esetén gátolja az input kimenetet. Minden input TTL kompatibilis.

Az érzékelő ellenállás, nem huzaltekercses típusú, kötelezően földelni kell a Vs negatív pólusa közelében. Ez a GND közelében van az IC-n. Minden bemenetnek a lehető legrövidebb úton kell csatlakoznia a vezérlőjel forrásához. Be és kikapcsolás előtt az En lábnak alacsony, tiltó állapotban kell lennie. A kimeneti feszültség használható arra, hogy szabályozza a kimeneti áram értékét az amplitúdó levágásával vagy túláram elleni védelmet lehet biztosítani, a kimenetek alacsonyra állításával.

A fék funkcióhoz ( gyors motorleállítás ) a maximális névleges teljesítmény nem lehet több mint 2 amper. Ha az ismétlődő áram maximuma több mint 2 amper, akkor párhuzamos konfiguráció is használható ( #7. ábra# ).

Külső diódákból álló hídra van szükség, ha induktív terheléseket hajtanak végre és az IC bemeneteire érkező jelek szaggatottak. ( Shottky diódákat célszerű használni. ) Ez a megoldás 3 Amperig képes működni egyenáramú üzemmódban és 3,5 A-ig ismétlődő csúcsáram üzemmódban.

**Guinness Mini DV Y2000 Webcam:** Egy apró méretű webkamera. Feladata a robot előtt található QR kódok beolvasása. A felbontása alacsony 600 x 480 másodpercenként 30 képkockát továbbít, AVI videó formátumot használ a videóhoz. A videó Ezek a paraméterek ideálisak, mert az alacsony képminőség miatt könnyebb feldolgozni a képkockákat. A készített képeket a szoftver JPG formátumban menti el a háttértárra a megfelelő utasítás beolvasása után. Egy Mini USB - USB kábelen keresztül történik a kommunikáció.

Egyes modellekben egy apró akkumulátor is található. A jelenlegi projektnél a tápellátást az USB port biztosítja. Képes egy max 32Gb-os SD kártyára rögzíteni, de ez a funkció nincs kihasználva. Méretei meglehetősen kicsik 26mm x 27mm x 26mm. Ez megkönnyíti a beszerelést.

(#kép a kicsi kameráról#)

**AVI ( Audio Video Interleave ):** Jelentése Audio Video Összefésülés. Egy fájlformátum, melyet a Microsoft 1992-ben mutatott be. Feladata a hang és a videó adatok egy csomagban való tárolása. Jelen esetben csak a kép továbbítódik.

Egy speciális esete a RIFF formátumnak ( Resource Interchange File Format - erőforrás-cserélő fájlformátum) mely a fájl adatait tömbökbe "chunk"-okba darabolja.

A RIFF formátumon belül az AVI egyetlen tömböt alkot. Ez két kötelező és egy nem kötelező altömbre van bontva.:

1. Az első altömb a "hdrl" címkével van ellátva. Ez a fájl fejléce és a metadatokat tartalmazza ( adat az adatról). Információt találunk itt többek között a kép szélességéről és magasságáról, a képkockák számáról.
2. A második altömb a "movi" címkével van jelölve. Ez a kép és hangadatokat tartalmazza, ezek alkotják az AVI videót.
3. A nem kötelező altömb az "idxl" címke jelöli, feladat a helymeghatározás a fájlon belül.

A RIFF formátumon keresztül a "movi" adatai dekódolhatóak és kódolhatóak egy modul segítségével. Ez a modul a kodek. Feladata a nyers adat a "movi" tömbön belül használt adatra fordítása. Ezért az AVI a médiatartalmat gyakorlatilag bármilyen tömörítőeljárásnak megfelelően tartalmazhatja, feltéve, hogy a tömörítési eljárás megfelel valamely sémának, de tömörítetlenül is tartalmazhatja ( például: Full Frames, Intel Real Time Video, QPEG, MPEG-4 ). A jelenlegi egyik legnépszerűbb megoldás az, ha a videót DivX-be a hangot Mp3-ba tömörítjük.

**JPEG (Joint Photographic Experts Group):** Képek tárolására alkalmas fájlformátum. A JPEG formátumot 1992-ben fogadták el, mint különböző képtömörítési eljárásokat leíró normát. A kép tartalmát veszteségesen tömöríti. A szabvány tartalmaz egy veszteségmentes tömörítést is de ezt nem használják. A tömörítés ugyan információveszteséggel jár, de egy 10-100x-osan tömörített kép a visszaállítása után is élvezhető marad. Fényképek és rajzok tárolására alkalmas első sorban. A Grafikonok és egyéb hirtelen színátmenetes ábrákhoz a PNG a megfelelő formátum.

A módszer különböző színmélységeket képes kezelni, használható szekvenciális és progresszív módon is:

szekvenciális: soronként történik a tömörítés.

progresszív: először durva formákat, majd a finomabb részleteket dolgozza fel.

JPEG-ben nem a képpontok tárolódnak le, hanem a képet transzformálják a frekvencia-tartományban DCT-vel (diszkrét koszinusz transzformáció).

Továbbfejlesztett változata a JPEG2000, mely a DCT helyett Wavelet transzformációt használ. Ez a 8x8 pixeles elemi blokkok határán jelent nagy különbséget.

A szabvány ugyan a tömörítést leírja de nem határozza meg a tárolási módokat. A képeket többnyire a JFIF ((JPEG File Interchange Format) ) formátumban tárolják. A kép feldolgozása többlépcsős:

1. Színtérszámítás RGB színtérből YCbCr színtérbe CCIR601 szerint
2. Cb és Cr színkomponensek szűrése és letapogatása.
3. 8x8-as blokkokra történő felosztás majd a blokkok diszkrét koszinusz transzformációja
4. Kvantálás
5. Átrendezés
6. Entrópiakódolás

Az 1,2,3,4-es lépéseknél képződik a veszteség. Ezt az emberi szem egészen a 1,5 - 2 bit/pixel adatmennyiségig nem látja, 0,3bit/pixel alatt a kép használhatatlan., mivel a tömörítés során olyan mintázatok kerülnek a képre amelyek korábban nem voltak ott, blokkok képződnek színek jelennek meg és tűnnek el, szürkül a kép a JPEG2000 ezeket a hibákat orvosolta.

Ha a forrásfájl 24-bit-RGB fájl, akkor 12-15-szörös tömörítésnél még szabad szemmel nem látható a veszteség, 35-ig még jó minőségűnek nevezhető a kép. Ezek a számok függenek a kép tartalmától is. Ha sok az apró részlet a képen ezek elveszhetnek a tömörítési eljárás során.

**Impulzusszélesség-moduláció (PWM):** (pulse-width modulation). Egy, az elektromos eszközök vezérléséhez használt széles körben elterjedt technológia. A korszerű teljesítményelektronika tett a gyakorlatban is használhatóvá.

A fogyasztóba táplált áramot, és az átlagos feszültséget úgy állítják be, hogy azt gyors ütemben be és kikapcsolják. Minél nagyobb a bekapcsolt állapot aránya a kikapcsolthoz képest a fogyasztó annál nagyobb teljesítményt tud felvenni.

A be és kikapcsolások frekvenciájának sokkal nagyobbnak kell lennie mint az a frekvencia ami még hatással lehet a fogyasztóra. Egy lámpa fényerő szabályozásánál ez a frekvencia jellemzően 120Hz egy motorvezérlő esetében néhány kHz-től pár száz kHz-ig terjedhet.

(#PWM példakép#)

A vonalfeszültséget (kék) a moduláció impulzusok sorozatává alakítja. Ez, szinuszos hullámformát indukál a fogyasztó áramkörében.

**ff-180sh DC motor:** Az alapként használt távirányítós autó gyári motorja. Az olcsó kínai eszközökben ez a típus vagy valamely klónja eléggé elterjedt, de a nevükön kívül az összes főbb paraméterük megegyezik. Hasonló változatok még az FF-180PA, FF-180SH, FF-180PH. A változatoktól függően a teljesítménye 1 - 2 wattól majdnem 20 watt közötti lehet. Ez függ a típushoz rendelt optimális feszültségtől. A fordulatszám is akár 28 000 fordulat / perc is lehet, de ennél a projektnél jelentősen kisebb. A motor fordulatszáma ugyanis egy áttételezéssel le van csökkentve, így a nyomaték növekedik.

A motor 3 Volt egyenárammal üzemel, szénkefés felépítésű. A legtöbb kis méretű egyenáramú villanymotor szénkefét használ, mert olcsó az előállítása és megbízhatóan üzemel. A motor két részből épül fel egy álló és egy forgórészből. Az állórész jelen esetben a házhoz van rögzítve ami csavarokkal kapcsolódik az autó vázához. A forgórész pedig maga a tengely. Egy lassító áttételen keresztül kapcsolódik a hátsó hajtott tengelyekhez. Az állórészben un.:egyenáramú gerjesztőtekercsek helyezkednek el amelyek 2 félkör-ív szerűen fogják közre a tengelyt. A forgórészen egyenáramú tekercselés található. A tekercselések végkivezetései a kommutátorra vannak kötve és a szénkeféken keresztül kapnak áramot a megfelelő időpontokban. A forgórész tekercseit úgy kell megtáplálni, hogy az általa keltett mágneses mező merőleges legyen az állórészben található mágneses térre. Így lehet a maximális hatásfokot elérni, mert a mágneses erő vonzó, illetve taszító hatása ilyenkor a legnagyobb.

A forgórész hornyolt felépítésű, a hornyokban helyezkednek el az egyenáramú tekercsek. A tengely tekercseibe a kommutátoron keresztül annyi ponton lép be és ki az áram ahány szénkefe zárja azt áramkört.

A forgórész által gerjesztett mágneses mezőt mindig a tengely szögsebességének megegyező sebességgel kell forgatni, csak azzal ellentétes irányban. A kommutátor feladata hogy a forgórész mágneses mezője az állórész mezőjéhez képest ne forduljon el. Ezt az állapotot közelítőleg tudja tartani úgy, hogy az megfelelő tekercsekben megfordítja az áram irányát, ha az áthalad egy a hornyokban található tekercs mágnesen pólushatárán. Ezért a forgórész-mezője maximum egy tekercsosztásnyit tud eltolódni, majd az ellentétes áramirány miatt visszakerül az eredeti helyére.

(#forrás linkje feljegyezve#)

**PC Java Szerver:** Feladata a Raspberry-n és az Androidon futó kliensek összekötése. Ezt socket kapcsolattal teszi lehetővé. A fejlesztés Windows alatt történt de a Linuxon történő futtatás is lehetséges. A Java szerver és egy kezdeti fázisban maradt PC kliens egyszerre készültek. A Java nyelv azért volt előnyös választás mert így a fejlesztés menete nagyon hasonló lehetett az Androidos eszközön, a Raspberry-n és a Windows vagy Linux operációs rendszert futtató PC-n.

A csatlakozott kliensektől érkező üzeneteket továbbítja az üzenetben elhelyezett címzettnek (ha van ilyen). Az üzeneteket nem tárolja, így ha az egyik kliens offline csak az utolsó üzenetet kapja meg a csatlakozásakor. A projekt jellegéből adódóan nem is szükséges az üzeneteket tárolni, de a megvalósítás lehetővé teszi, hogy némi módosítás után rendelkezzen az üzenet tárolásának lehetőségével. Így, akár egy chatprogram szervere is lehet. Egy ilyen működést a kliensek is támogatnak a szükséges módosítások elvégzése után.

A Java nyelvben választása öt fő okra vezethető vissza:

1. A párhuzamos szálakkal történő programozást egyszerűen lehet benne megvalósítani
2. A platformfüggetlensége miatt az összes szükséges operációs rendszert támogatja
3. A nyelv objektumorientált. A részegységek egységbezárása miatt és a logikai elemek elkülönítése miatt volt fontos.
4. A Socketen keresztül történő kommunikáció megvalósítására könnyen használható eszközök állnak a rendelkezésre.
5. Egyszerűen készíthető grafikus vezérlőfelület (GUI) a korábban csak konzolos felületen karakteres megjelenítő helyett.

A párhuzamos szálkezeléssel segítségével lehet megvalósítani a kliensek elválasztását egymástól. Mivel két klienst kell egyszerre kezelni, ezért ha a megvalósítás jó, akkor nem igényel sok erőforrást. Ilyenkor a kliensekhez kötődő folyamatok a szerveren egymástól függetlenül futnak, egy esetleges hiba nem állítja le az egész programot.

A szálak egymással egy mindenki számára elérhető és módosítható objektumon keresztül kommunikálnak. Ha egy szál megszerzi a neki szükséges információt akkor törli azt, annak érdekében, hogy legközelebb már ne olvassa ki. Ha megvan az üzenet akkor továbbítja a kliensnek, vagy ha a szervernek szóló tartalom van benne akkor végrehajtja azt.

Az első változatban a megvalósítás nem volt tökéletes, így a szervernek igen magas lett a hardverigénye. Ennek oka a szálak kezelésében volt keresendő, valamint a szálak futása során elvégzett tevékenységekben. A futtatás nagy erőforrásigénye miatt a szerver több váratlan hibát is produkált:

1. Lefagyott
2. Eldobta a kapcsolatot a kliensekkel
3. Nem küldte el az üzenetet
4. Többször is elküldte az üzenetet
5. Váratlanul kilépett
6. A grafikus felület fagyott le, de a működés továbbra is hibamentes volt

Ezeket a hibákat a szálkezelés rendberakása után nem produkálta. A megoldás a szálak futásának időzítése és a szálak futásának szüneteltetése volt. Ha egy kliens üzenetet küld akkor azt a program továbbítja vagy végrehajtja majd a száll futása szünetel egy darabig. Ha egy kliens lecsatlakozik, vagy nem válaszol az ellenőrző üzenetre akkor a szál leáll. Korábban előfordult, hogy a lecsatlakozott kliens után a hozzá rendelt folyamat még tovább futott és pazarolta az erőforrásokat. Egy klienshez egy fogadó és egy üzenetküldő szál társul. Így minden felvett új kliens kettővel növeli a futó folyamatok számát.

A kliensek csatlakozásának ellenőrzését a szerver egy watchdog timer szerű működéssel oldja meg. Egységes időközönként üzenetet küld a klienseknek akik válaszolnak az üzenetre. Ha nincs válasz akkor megismétli a következő periódusban. Ha ekkor sem kap választ akkor az adott szálat leállítja. Ilyenkor a kliens újra csatlakozhat.

**watchdog timer ( felügyeletidőzítő ):** Beágyazott rendszerekben és más számítógépes vezérlésű eszközökben használják ahol a felhasználó nem fér hozzá az eszközhöz, vagy nem tud időben reagálni a fellépő hibára. Egy ilyen rendszer nem függhet a felhasználótól ha valamilyen probléma adódik önállóan kell arra megoldást találnia. A módszert széles körben használják az űrszondákban, hiszen fizikailag azok is hozzáférhetetlenek.

Operációs rendszerekben is használják, egy időintervallum áll bizonyos műveletek végrehajtására. Ha a művelet nem végez az intervallumon belül akkor a rendszer rögzíti a hibaüzeneteket majd leállítja a folyamatot.

Számos konfigurációban előfordulnak, gyakran a mikrovezérlők is tartalmazzák. A számítógépekben egy a processzorhoz közeli chipben vagy egy bővítőkártyán helyezkedhet el. Futhat a processzorral azonos és különböző órajelen is.

**Java szerver felépítése:** A szerver 2 fő részegységből áll. Az egyik a működést megvalósító egység ( modell ) amely végzi a kliensekkel történő kommunikációt, a szálkezelést, kliensek ellenőrzését stb. A másik a grafikus felület ( GUI ) amelyen keresztül tájékoztatja a felhasználót az aktuálisan futó folyamatokról, csatlakozott felhasználókról, kiírja a hibaüzeneteket, mutatja, hogy hány szál fut aktuálisan. Egy szövegdobozban kiírja a kliensek közötti üzeneteket. Ha egy chat program szervereként üzemelne ezt a privát üzenetek esetén nem tenné, vagy a kliensek valamilyen kódolt formában üzennének egymásnak és csak a hash kódokat jelenítené meg.

A kliensek csatlakozásuk után elküldik a saját felhasználónevüket, ezzel lehet majd őket azonosítani a továbbiakban. A később küldött üzeneteknek az első részben tartalmazniuk kell az adott üzenet címzettjét. Ha a kapcsolódás sikeres volt a szerver indít egy küldő szálat és egy fogadó szálat. Ha egy kliens üzenetet küld a fogadó szál kiolvassa belőle a címzettet és a megfelelő küldő oldali szál által elérhető változóban letárolja. Ha a küldő szál észleli, hogy a küldendő üzenetet tartalmazó string nem üres továbbítja az üzenetet. Továbbítás után törli a string tartalmát, ezzel elkerülve, hogy folyamatosan az utolsó kapott üzenetet továbbítsa. A kliens addig hatja végre az utoljára kapott utasítást amíg más nem kap vagy elér egy bizonyos végállapotot pl.: kerekek szélső értékig elfordultak.

Ha a szerver olyan üzenetet fogad amelynek a címzettje nem létezik, akkor figyelmen kívül hagyja azt. Mivel 2 eszköz kommunikál ezért nem szükséges visszajelezni, hogy az adott felhasználó nem létezik, de egy chat szolgáltatást megvalósító szervernél ezt a lépést célszerű megtenni.

A szerver az indulást követően elvégzi saját magán a szükséges beállításokat. Nyit egy Server Socketet a 10000-res portra. Elindítja a csatlakozott felhasználókat figyelő szálat és a futó folyamatokat számoló szálat is. A grafikus felületen a szövegdobozok formázását html tag-ekkel lehet megvalósítani, így ezeket is be kell állítani.

Kliens csatlakozása után a hozzá rendelt fogadó és küldő szálat is elindítja. Jelzi a felhasználónak a csatlakozást, és ha megvan a kliens felhasználóneve akkor kiírja a megfelelő, csatlakozott felhasználókat listázó szövegdobozba.

(# kép a grafikus felületről#)