

УВОД

Системите за търсене и проследяване на цели са предназначени за идентифициране, локализиране, проследяване и събиране на информация за конкретни обекти в дадена среда. Те използват комбинация от сензори и актуатори, за да сканират пространството около тях. След като цел бъде идентифицирана, алгоритми за проследяване, предвиждат бъдещите ѝ движения и ефективно следват траекторията ѝ, предоставяйки данни и ситуациянна осведоменост на оператора. Тези системи непрекъснато сканират заобикалящата ги среда, като се стремят да идентифицират и поддържат контакт със целите около тях.

Тези системи играят жизненоважна роля в различни области, вариращи от приложения в областта на от branата и сигурността до приложения като автономни превозни средства, индустрисална автоматизация и роботика. Тяхната способност за прецизно идентифициране и проследяване на обекти в реално време допринася значително за повишаване на оперативната ефективност, безопасността и цялостното функциониране на системите в множество области, както и за осигуряването на ниво на ситуациянна осведоменост, което не може да бъде постигнато от други видове системи.

ПЪРВА ГЛАВА

1.1 Проучване на съществуващи системи за сканиране и проследяване на цели

1.1.1 Комерсиални системи

Dedrone е фирма специализираща се в създаването на решения за откриване на летящи обекти. Едно от техните предлагани решения е оптична триста и шейсет градусова камера (фиг. 1.1) за откриване и проследяване на цели в реално време. Тя е интегрирана в платформата Dedrone, която е проектирана да осигурява цялостна сигурност на въздушното пространство. Камерата е оборудвана с функции за завъртане, накланяне и мащабиране, което позволява наблюдение на голяма площ въздушно пространство с едно устройство. Тя има нормален обхват за дронове до един километър, а в режим на нощното (с помощта на инфрачервен светодиод) достига до четиристотин метра. Тя разполага и с тридесеткратно оптично мащабиране на картина, което улеснява откриването и идентифицирането на малки дронове на голямо разстояние.



Фигура 1.1: Dedrone PTZ камера

Камерата осигурява висока разделителна способност от хиляда и осемдесет на хиляда деветстотин и двадесет пиксела. Една от ключовите характеристики на продукта е интеграцията със софтуера DedroneTracker.AI. Този софтуер разполага с интелигентна възможност за анализ на видео, която открива и локализира дронове в реално време. Функцията за автоматизирано проследяване непрекъснато проследява дрона, като предоставя на операторите актуална информация за местоположението му във въздушно пространство.

GroundAware GA3360 (фиг. 1.2) на Aerial Armor е радарна система с пълно покритие на хоризонта, предназначена за наблюдение както на земни, така и на въздушни цели. GA3360 може да открива, проследява, класифицира и реагира на заплахи, причинени от хора, животни, наземни превозни средства, самолети и дронове. Той измерва височината на целите и позволява на системата да алармира само за целите, които предизвикват беспокойство. GA3360 работи в S-обхвата (3,156 до 3,25 GHz) и има регулируема разделителна способност на обхвата от десет до двайсет метра. Той има обхват на откриване на хора от два километра, обхват на откриване на превозни средства от три километра и обхват на откриване на дронове от един цяло и пет километра. Обхватът на откриване на самолети също е три километра. Системата е с тегло по-малко от четиристотин килограма и размери 22,5 на 21,3 инча. Една от ключовите характеристики на GA3360 е възможността за интегриране със собствения софтуер за откриване на дронове на Aerial Armor. Този софтуер автоматично насочва камерата към зоната на активност, идентифицирана от радара, за визуално потвърждение. Това значително подобрява точността на откриване и намалява фалшивите сигнали.



Фигура 1.2: GA3360 радар

Aeron Ranger (фиг. 1.3), произведен от Aerial Armor е хоризонтално и вертикално насочваема камера, която използва неохлаждан LWIR (дълговълнов инфрачервен) сензор с разделителна способност 640 на 480 пиксела. Хоризонталното зрително поле варира от 25,4 (широкоъгълно) до 4,1 градуса при тридесеткратно оптично увеличение. Той има рейтинг за ниво на защита от вредното въздействие на околната среда IP67 и може да се използва в тежки и предизвикателни приложения, като например в морския сектор, в сферата на граничната сигурност и в инсталации за монтиране на превозни средства. Обхватът на откриване на термокамерата е до 3,29 км за хора и до 10,1 км за превозни средства.



Фигура 1.3: Aeron Ranger радар

DJI AeroScope G8 (фиг. 1.4) и **G16** са решения за откриване на дронове, които предлагат цялостна защита срещу летящи цели. G8, преносима система, монтирана на статив, се отличава с 90° покритие на сигнала на антена, с два антенни модула, работещи на честоти 2,4 GHz и 5,8 GHz. С по-дълги обхвати на откриване и интегрирано географско ограждане G8 моделът осигурява надеждни възможности за защита.

За разлика от него стационарното устройство G16 се отличава с рейтинг за устойчивост на прах и вода IP65 и по-широк температурен диапазон на работа, 360° покритие и по-голям обхват на откриване на цели. И двата модела имат сходни спецификации, включително, работни честоти и входно съпротивление, но G16 е специално пригоден за стационарна употреба, като осигурява гъвкави възможности за конфигурация, както и за непрекъснато наблюдение на открито.



Фигура 1.4: DJI AeroScope радари G8 и G16

1.1.2 Военни системи

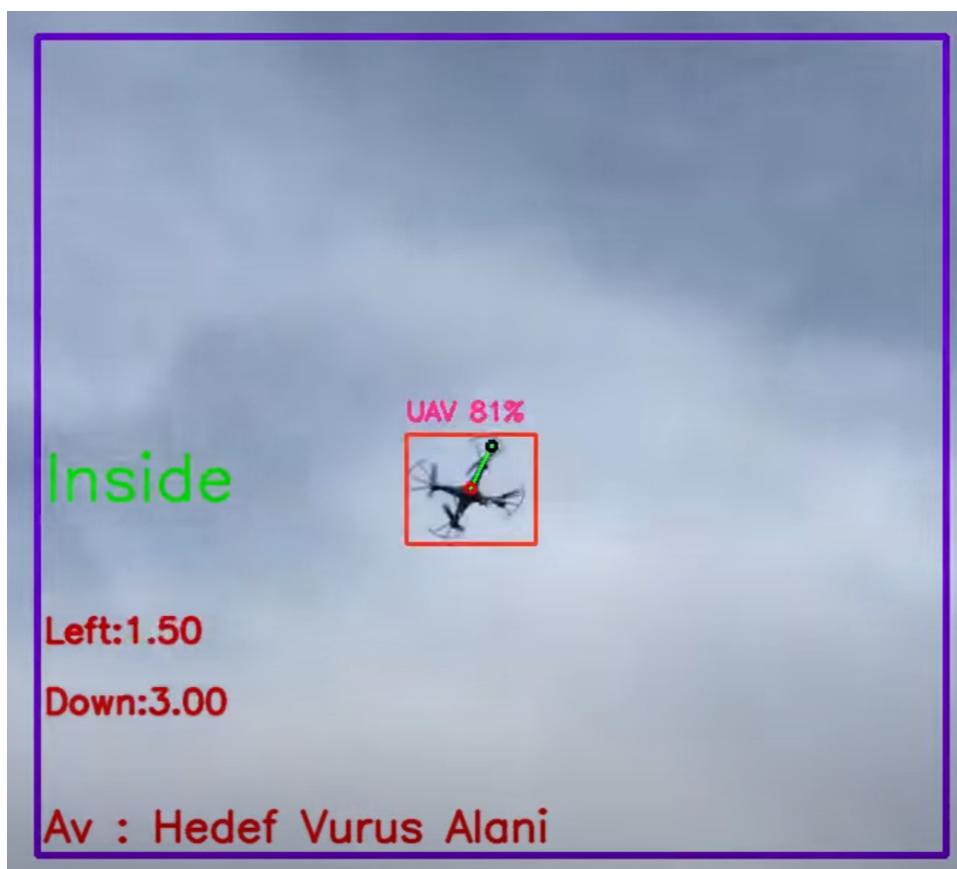
AN/MPQ-64 Sentinel (фиг. 1.5) е електронно управляема импулсно-доплерова радарна система в работеща в честотния диапазон X на сантиметровите дължини на вълните, произвеждана от Raytheon Missiles and Defense, която се използва за предупреждение и насочване на оръжия за противовъздушна обрана с малък обсег. Тя използва въртяща се платформа с висока скорост на сканиране (30 оборота на минута), за да осигури 360-градусово азимутално покритие и височинен обхват от -10° до $+55^{\circ}$. Радарът е проектиран с висока устойчивост на електронни средства за противодействие, автоматично придобива, проследява, класифицира, идентифицира и докладва цели на голяма и малка височина, включително крилати ракети, безпилотни летателни апарати, както и ротационни и фиксирани самолети. Идентифицирането на приятелски въздухоплавателни средства се подпомага от запитване за идентификация на приятел или враг (IFF). Обхватът на системата е до 40 километра за цели с голямо радарно сечение. Осигурява 360° азимутално покритие и височинен обхват от -10° до $+55^{\circ}$. Системата може да бъде разположена заедно с 10 kW 400 Hz 115/200 VDC генератор. Радарната система може също така да разпространява данните си по мрежа за предаване на данни.



Фигура 1.5: AN/MPQ-64 Sentinel военен радар

1.1.3 Непрофесионални системи

Системите за проследяване на цели, създадени от любители най-често работят с помощта на камера, като разчитат на техники за компютърно зрение за откриване и проследяване на обекти в рамките на видеопоток (фиг. 1.6). Тези програми могат да бъдат реализирани с помощта на Python и OpenCV, популярна библиотека за компютърно зрение с отворен код. Производителността може да бъде силно зависима от възможностите на камерата и изчислителната мощ на компютъра, на който работи програмата. Вероятността на откриване и точността и скоростта на проследяване варира в зависимост от фактори като условия на осветление, метеорологични условия, външния вид на обекта и качеството на подаваният от камерата видеопоток. Възможностите за откриване и проследяване обикновено са съобразени с конкретни видове обекти или класове, обучени по време на фазата на разработване, което ограничава гъвкавостта на програмата. В любителските реализации може да липсват функции като проследяване на множество обекти, възможност за изчисляване на информация като вектор на движение, разстояние, размер, надморска височина, скорост и други. Въпреки че програмите за проследяване на цели, създадени с помощта на OpenCV, са лесни и евтини за реализиране и обслужват основни нужди за проследяване, те не отговарят на изискванията за производителност, точност и надеждност нужни в повечето ситуации.



Фигура 1.6: Любителски софтуер за проследяване на цели

1.1.4 Сравнение и анализ на съществуващи системи

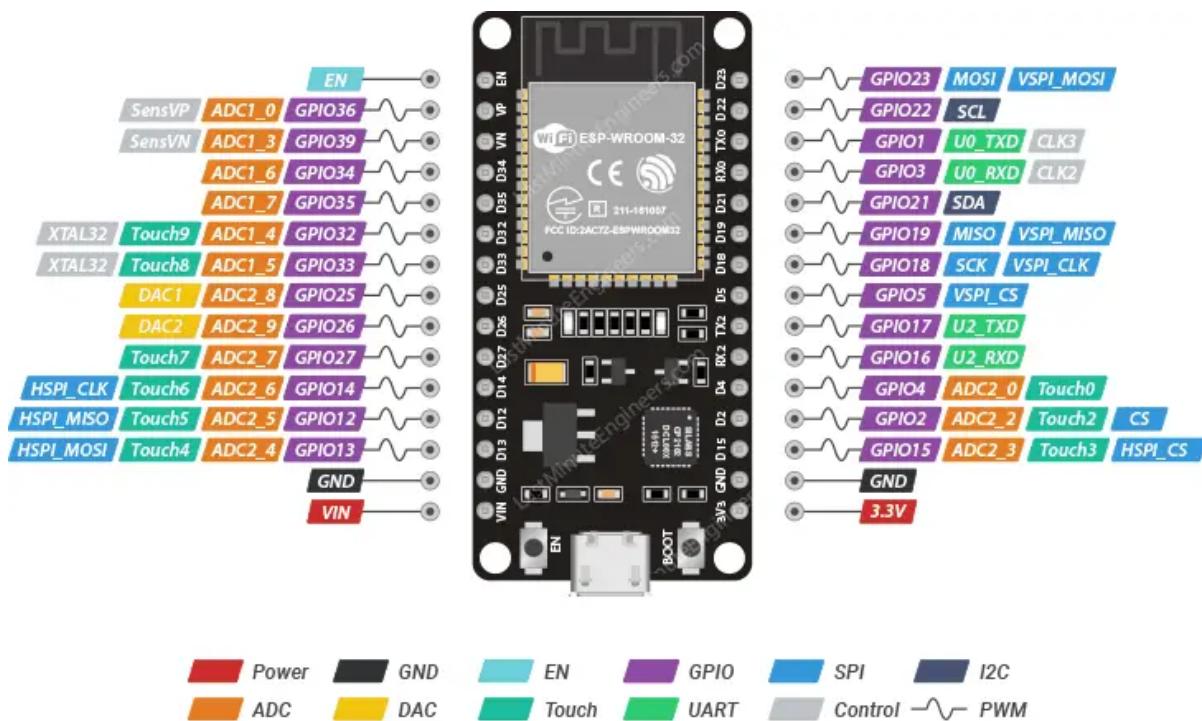
Системите за проследяване на цели имат различни възможности и пригодност за различни приложения. Любителските системи, използващи камери и техники за компютърно зрение, като например тези, използващи Python и OpenCV, предлагат икономически ефективни решения, но са ограничени от фактори като качество на камерата и изчислителна мощност. Въпреки че тези системи осигуряват основни функции за проследяване, често им липсва устойчивост и гъвкавост, особено при неблагоприятни условия на околната среда и сценарии за проследяване на множество обекти. За разлика от тях, военните системи като радара AN/MPQ-64 Sentinel предлагат усъвършенствани функции като високоскоростно сканиране, цялостна класификация на целите и устойчивост на електронни контрамерки. По подобен начин, комерсиално налични продукти като DJI AeroScope и Aeron Ranger предлагат специализирани възможности, пригодени за откриване на дронове, с характеристики като по-широко покритие, по-дълги диапазони на откриване и интеграция със специални платформи за наблюдение. GroundAware GA3360 на Aerial Armor допълнително разширява възможностите чрез интегриране на 3D радарна технология със софтуер за визуално потвърждение за по-голяма точност. Решението за оптична камера на Dedrone, интегрирано с усъвършенстван софтуер, демонстрира важността на откриването и проследяването в реално време, като предлага цялостна сигурност на въздушното пространство с интелигентни функции за анализ. Като цяло, докато любителските системи обслужват основните нужди от проследяване, търговските и специализираните решения осигуряват превъзходна производителност, точност и надеждност за разнообразни приложения за наблюдение, но тези системи струват около полин милион лева, за да се инсталират в определена зона и само военният решения имат точност на измерване, с което да позволят изчиляване на огнево решение срещу малки цели като дронове.

1.2 Устройства избрани за дипломния проект

1.2.1 Микроконтролер

Микроконтролерът ESP32

ESP32 Devkit V1 (фиг. 1.7) е мощен микроконтролерен модул, който се използва широко във вградените системи и приложенията на интернет на нещата благодарение на своята гъвкавост, производителност и интегрирани безжични възможности. Разработен от Espressif Systems, ESP32 служи като наследник на ESP8266, предлагайки значителни подобрения по отношение на изчислителната мощност, свързаността и функционалността. В основата на ESP32 е интегриран двуядрен микропроцесор Xtensa LX6, който осигурява тактова честота до 240 MHz. Тази двуядрена архитектура подобрява производителността, като позволява едновременното изпълнение на задачи. ESP32 включва и богат набор от периферни устройства и интерфейси, което го прави изключително адаптивен за разнообразни проекти за вграждане.



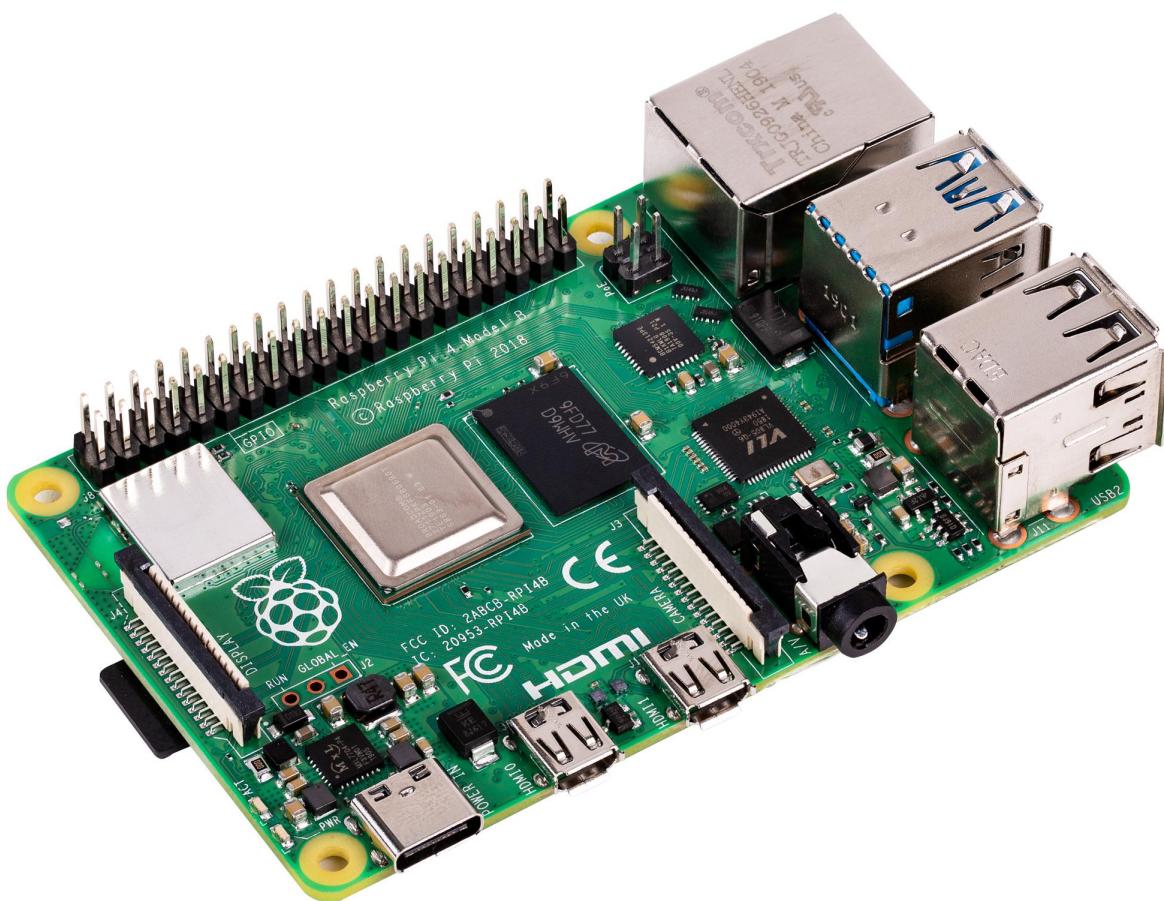
Фигура 1.7: ESP32 Devkit V1 pinout

Той разполага с множество GPIO (Вход/изход с общо предназначение), SPI (Сериен периферен интерфейс), I2C (Inter-Integrated Circuit), UART (универсален асинхронен приемник-предавател) и ШИМ (Широчинно импулсна модулация) изводи, улесняващи връзката с различни сензори, изпълнителни механизми и външни устройства. Освен това ESP32 включва вградени функции за сигурност, като криптиране на флаш памет, което гарантира целостта на данните и повишава цялостната сигурност на устройството в което е използвано. Този микроконтролерен модул се допълва от широка поддръжка за разработка и стабилна софтуерна екосистема. Той може да бъде програмиран с помощта на Arduino IDE (интегрирана среда за разработка) или Espressif IDF (IoT Development Framework), като предлага гъвкавост и лекота на разработката както за начинаещи, така и за опитни разработчици. Универсалността на този микроконтролер го прави подходящ за широк спектър от вградени проекти, включително домашна автоматизация, интелигентни устройства, индустриална автоматизация, носими устройства и др. Ниската му консумация на енергия, съчетана с възможностите му за обработка и безжичната свързаност, го прави идеален избор за приложения в множество вградени системи.

Микрокомпютърът Raspberry Pi 4

Разработен от фондация Raspberry Pi, този компютър с размерите на кредитна карта служи като универсална платформа, подходяща за широк спектър от приложения - от образователни проекти до промишлени приложения и творения на ентузиастите "Направи си сам". Raspberry Pi 4 използва четириядрен процесор ARM Cortex-A72, работещ на честота до 1,5 GHz, който осигурява значително по-висока производителност в сравнение с предишните модели. Тази увеличена изчислителна мощност, съчетана с възможности за различни конфигурации на оперативната памет (от 2 GB до 8 GB), значително подобрява възможностите за многозадачна работа, изчислителната производителност и цялостната бързина на реакция.

Raspberry Pi 4 поддържа съвместимост с предишните модели, като запазва 40-пиновия GPIO (General Purpose Input/Output) хедър, което позволява безпроблемна интеграция с широка гама сензори, дисплеи, двигатели и други електронни компоненти. Този GPIO хедър, съчетан с интерфейси като I2C, SPI и UART, позволява широко хардуерно взаимодействие, което го прави популярен избор както за любители, така и за професионалисти.



Фигура 1.8: Raspberry Pi 4

Освен това Raspberry Pi 4 поддържа различни операционни системи, включително официалната Raspberry Pi OS (бивша Raspbian), Ubuntu и няколко други дистрибуции на Linux. Тази гъвкавост позволява на потребителите да приспособят системата към специфичните си нужди, независимо дали става въпрос за програмиране, уеб сърфиране, медиен стрийминг или хостинг на сървъри.

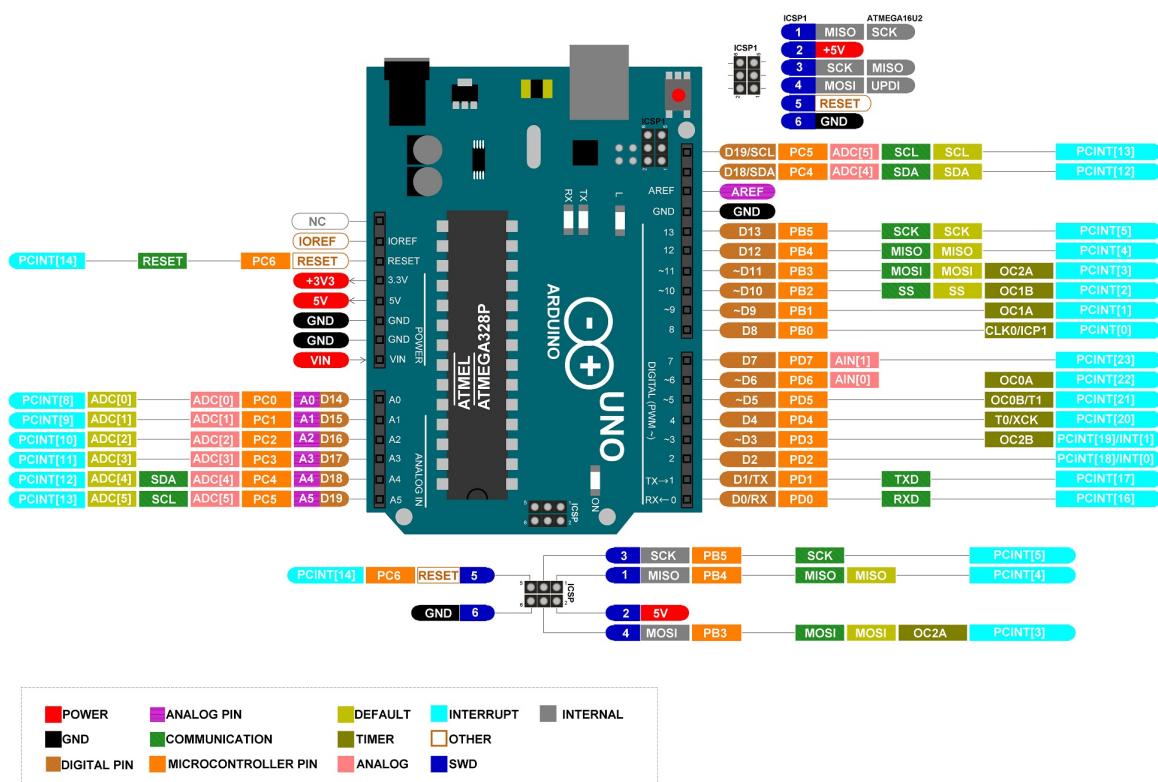
Микроконтролерът Arduino Uno R3

В основата си Arduino Uno (фиг. 1.9) се захранва от 8-битов микроконтролер Atmega328P с тактова честота 16 MHz, който предлага баланс между производителност и лекота на използване. Този микроконтролер осигурява 32KB флаш памет, 2KB SRAM и 1KB EEPROM, което му позволява да се справя ефективно с широк спектър от задачи.



Фигура 1.9: Arduino Uno R3

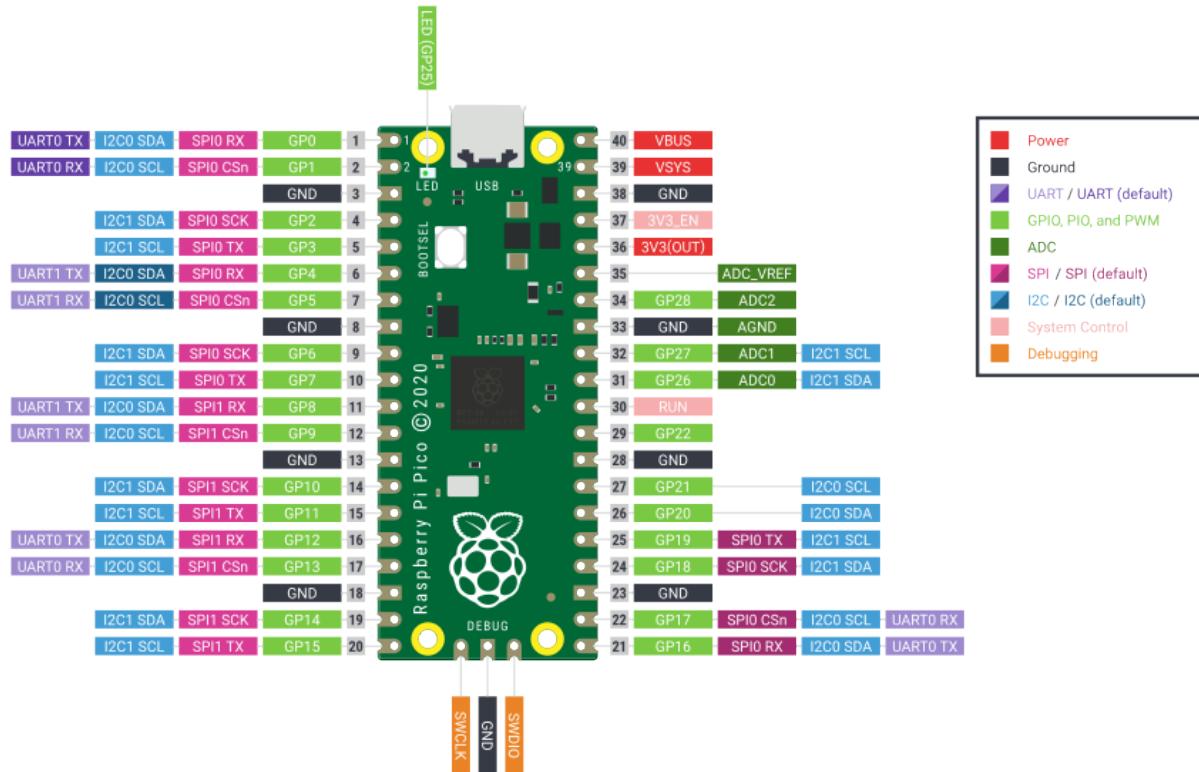
Микроконтролерът включва прости I/O (вход/изход) интерфейси с 14 цифрови извода, сред които 6 извода могат да се използват като PWM (широкочинно-импулсна модулация) пинове. Освен това той предлага 6 аналогови входни извода, позволяващи свързване към различни аналогови сензори и устройства. Платката включва също така захранващи изводи, UART (универсален асинхронен приемник-предавател) за серийна комуникация, SPI (сериен периферен интерфейс) и I2C за комуникация с други устройства. Освен това Arduino Uno може да се захранва чрез USB връзка или външен източник на захранване, което осигурява гъвкавост при различни сценарии на използване (фиг. 1.10).



Фигура 1.10: Arduino Uno R3 Pinout

Микроконтролерът Raspberry Pi Pico

Raspberry Pi Pico (фиг. 1.11) е компактна микроконтролерна платка, предназначена за широк спектър от вградени проекти. Той предлага рентабилно, но мощно решение както за ентузиасти, любители, така и за професионалисти. Той използва микроконтролерния чип RP2040, който е проектиран от самите Raspberry Pi. Това е двуядрен ARM Cortex-M0+ процесор с тактова честота 133MHz и осигурява процесорна мощност за различни задачи, като същевременно остава енергийно ефективен. Pico е оборудван с 264KB SRAM и 2MB вградена флаш памет, осигурявайки достатъчно място за съхранение на програми и манипулиране на данни. Освен това поддържа външно съхранение чрез microSD карти. Raspberry Pi Pico предлага набор от опции за периферно свързване. Той включва 26 GPIO пина за взаимодействие със сензори, задвижващи механизми и други външни устройства. Тези щифтове поддържат различни комуникационни протоколи, което го прави универсален за широк спектър от приложения. Raspberry Pi Pico се поддържа от обширна екосистема за разработка, включително официална документация, софтуерни библиотеки и форуми на общността. Може да се програмира с помощта на различни езици за програмиране, включително MicroPython, C/C++ и CircuitPython.



Фигура 1.11: Raspberry Pi Pico pinout

Сравнение и избор на микроконтролер

Сравняването на микроконтролерите Raspberry Pi Pico, Arduino Uno, Raspberry Pi 4 и ESP32 разкрива различни характеристики и пригодност за различни вградени приложения. Raspberry Pi Pico, задвижван от микроконтролерния чип RP2040, предлага баланс между ценова ефективност и производителност с голям обем памет, GPIO изводи и поддръжка на множество езици за програмиране. Arduino Uno, със своя микроконтролер Atmega328P, осигурява простота и лекота на използване, подходящ за широк спектър от задачи, макар и с ограничен размер на паметта и изчислителна мощ в сравнение с по-усъвършенстваните варианти. Raspberry Pi 4 се отличава с четириядрения си процесор, разширениите възможности за оперативна памет и обширния GPIO хедър, което го прави подходящ за многозадачна работа и разнообразни хардуерни взаимодействия. Въпреки това за избраното приложение ESP32 се очертава като оптimalен избор поради по-големия размер на паметта, скоростта на обработка, компактния физически размер, разнообразната поддръжка на логически нива и конкурентната цена. Със своя двуядрен процесор Xtensa LX6, работещ на честота до 240 MHz, ESP32 предлага повишена изчислителна мощност за едновременно изпълнение на задачи. Освен това ESP32 включва вградени функции за сигурност, като криптиране на флеш паметта, което гарантира целостта на данните и сигурността на устройството. Поддържан от стабилна екосистема за разработка и съвместим с популярни среди за разработка като Arduino IDE и Espressif IDF, ESP32 осигурява гъвкавост и лекота на разработката както за начинаещи, така и за опитни разработчици. Освен това опциите за безжична връзка позволяват бъдещо развитие на проекта.

1.2.2 Сензор за измерване на разстояние

Lidar Lite v3 (Фиг. 1.12) е компактен, високопроизводителен оптичен сензор за измерване на разстояния. Той работи при 5 VDC (максимум 6 V) с пикова мощност от 1,3 W. Сензорът има обхват от 1 до 40 м и разполага с излъчващ в краищата, 905 nm (1,3 W), еднолентов лазерен предавател, 8 м радиална дивергенция на лъча и оптична апертура от 12,5 mm. Цената му е 149,99 USD. Lidar Lite v3 комуникира чрез I2C и PWM и съществува библиотека Arduino за лесно свързване.



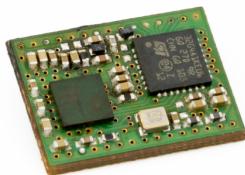
Фигура 1.12: LIDAR Lite v3

Lidar Lite v4 (Фиг. 1.13) е друг високопроизводителен оптичен сензор за измерване на разстояние от Garmin. Цената му е 74,95 USD и предлага обхват до 10 метра с разделителна способност 1 см. Сензорът комуникира чрез I2C и PWM и съществува библиотека Arduino за лесно свързване.



Фигура 1.13: LIDAR Lite v4

V-LD1 (Фиг. 1.14) е 61GHz FMCW радарен сензор за разстояние, който предлага прецизно измерване на разстоянието с точност до mm. Той може да измерва разстояния до 50 м в зависимост от средата и използвания обектив. Сензорът е със свръхмалък SMD форм-фактор (12 mm x 16 mm) и работи с едно 1,8V захранване. Изходните данни за разстоянието се предават чрез сериен UART интерфейс.



Фигура 1.14: V-LD1 радарен сензор за разстояние

Развойният комплект V-LD1 (Фиг. 1.15) е мощен инструмент за оценка на сензора V-LD1. Той включва сваляща се пластмасова леща, която фокусира радарния лъч до приблизително 8° до 8° , което е идеално за по-големи разстояния на откриване. Комплектът осигурява поточно предаване и визуализация на всички данни от сензора в реално време и поддържа запис и възпроизвеждане на данни. Цената на комплекта за оценка е 192,46 USD.



Фигура 1.15: V-LD1 развоен комплект с обектив

Сравнение, анализ и избор на микроконтролер

При сравнителния анализ на Lidar Lite v3, Lidar Lite v4, сензора за разстояние V-LD1 и комплекта за оценка на V-LD1, Lidar Lite v3 и v4 се оказаха предпочтитани за проекта. Основните фактори, които повлияха на това решение, бяха тяхната рентабилност и лесното им свързване с микроконтролер. Lidar Lite v3 и v4, чиято цена е съответно 149,99 USD и 74,95 USD, предлагат значително ценово предимство пред комплекта за оценка V-LD1, чиято цена е 192,46 USD. Тази ценова разлика е особено изразена, когато се разглеждат функционалностите, предлагани от всеки сензор, като моделите Lidar Lite осигуряват солиден набор от функции на по-ниска цена. Освен това сензорите Lidar Lite v3 и v4 комуникират чрез I2C и ШИМ и има съществуващи библиотеки за Arduino за лесно свързване, което ги прави по-достъпни за интегриране с микроконтролери. За този проект Lidar Lite v4 беше използван като доказателство за концепцията, за да се спестят разходи. Въпреки това, е възможна лесна замяна с Lidar Lite v3.

1.2.3 Мотори за насочване на сензора за измерване на разстояние

Серво мотор MG996R

Сервомоторът MG996R (Фиг. 1.16) е двигател с висок въртящ момент, предназначен за прецизно управление в проекти за роботика и автоматизация. С диапазон на въртящия момент от 10 kg/cm до 15 kg/cm (4,8 V до 6 V) той предлага висока изходна мощност, позволяваща стабилни движения и точност на позициониране. Неговият метален редуктор повишава издръжливостта и производителността, подходящ за различни приложения, изискващи надеждно и прецизно управление на движението. Освен това той работи в диапазона на напрежението от 4,8V до 7,2V, осигурявайки гъвкавост на опциите за захранване.

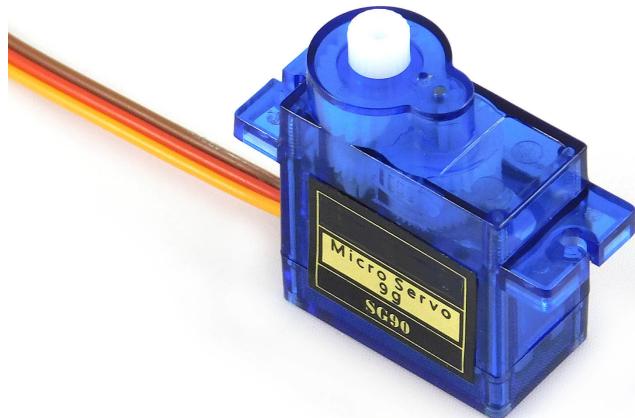


Фигура 1.16: MG996R стъпков мотор

Серво мотор SG90

Микросервото SG90 (Фиг. 1.17) е компактен и лек двигател, подходящ за малки проекти. Той предлага диапазон на въртящия момент от приблизително 1,8 kg/cm, което го прави идеален за приложения, изискващи прецизно, но не толкова взискателно управление на движението.

Работейки в диапазона на напрежението от 4,8 V до 6 V, този сервомотор е известен със своята достъпност, ниска консумация на енергия и гъвкавост, което го прави подходящ за любителски проекти, миниатюрна роботика и моделни приложения.



Фигура 1.17: SG90 стъпков мотор

Стъпков мотор NEMA17

Стъпковият двигател NEMA 17 (Фиг. 1.18) е широко използван тип стъпков двигател, известен със своята гъвкавост и надеждност в различни приложения за управление на движението. Наречен така по името на стандарта на Националната асоциация на производителите на електротехника (NEMA), който определя физическите му размери, той има рамка с квадратна форма и размери 42 на 42 mm. Той се характеризира със способността си да се движи на точни, инкрементални стъпки, което го прави подходящ за приложения, изискващи точно позициониране и управление. Този стъпков двигател обикновено има ъгъл на стъпката от 1,8 градуса на стъпка, въпреки че някои варианти могат да предлагат по-висока разделителна способност.



Фигура 1.18: NEMA17 стъпков мотор

Финален избор на мотори за насочване на сензора за измерване на разстояние

Сервомоторът MG996R бе избран заради надеждна работа и лесна интеграция в системата. Високият изходящ въртящ момент на MG996R и издържливото метално зъбно колело напълно отговарят на изискванията на проекта за прецизно и мощно управление на движението.

1.2.4 Дисплей и управление

Adafruit 3.5" TFT Touchscreen

3,5-инчовият TFT сензорен еcran на Adafruit (Фиг. 1.19) е популярен дисплей с 480x320 пиксела резолюция. Дисплеят има резистивен сензорен еcran, който позволява въвеждане на данни с докосване от потребителя. Той може да се управляват със стилус или пръст. Контролерът на дисплея е базиран на чипа ILI9341, който е често срещан контролер за TFT дисплеи. Това го прави съвместим с различни библиотеки и платформи, включително Arduino и ESP32. Използва SPI (Serial Peripheral Interface) интерфейс за комуникация с микроконтролери като ESP32. Дисплеят е снабден с LED подсветка, която осигурява равномерно осветяване на екрана, като яростта може да се контролира чрез софтуер, което позволява регулиране в зависимост от условията на околното осветление. Adafruit предоставя библиотеки и примерен код, за да улесни интеграцията с различни платформи. Физическите размери на модула са приблизително 85 mm x 55 mm x 10 mm (3,3"x 2,2"x 0,4"), което го прави компактен и подходящ за вграждане в проекти с ограничено пространство.

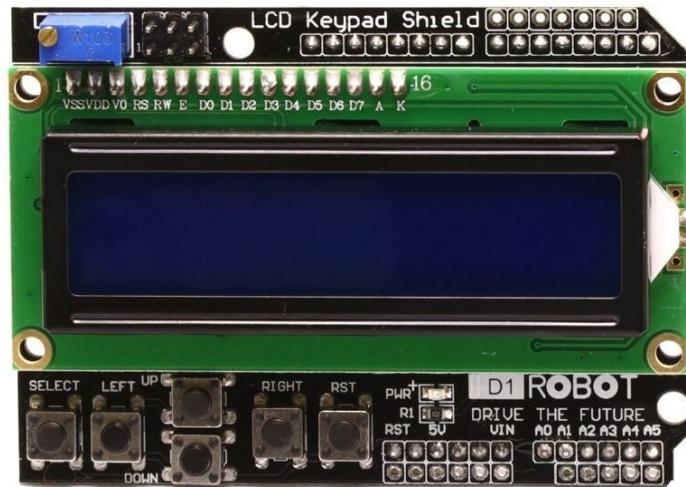


Фигура 1.19: Adafruit 3.5" TFT Touchscreen дисплей

DFRobot LCD Keypad

Модулът на DFRobot с LCD клавиатура (Фиг. 1.20) разполага с 16x2 символа LCD (течнонокристален дисплей). В допълнение към LCD дисплея модулът включва клавиатура с 5 бутона за интерфейс с потребителя. Те осигуряват прост интерфейс за взаимодействие с потребителя. Подсветката на дисплея може да се контролира чрез софтуер, за да се регулира яростта или да се включва/изключва при необходимост.

Въпреки че е предназначен основно за платки Arduino, дисплеят може да се използва и с други платформи за микроконтролери, които имат съвместимо разположение на изводите, като например ESP32. Библиотеки като LiquidCrystal предоставят лесни за използване функции за управление на LCD дисплея и четене на входни данни от клавиатурата. Модулът работи с 5 V DC.



Фигура 1.20: DFRobot LCD дисплей с клавиатура

Преносим компютър

Преносимите компютри (Фиг. 1.21), често наричани лаптопи или ноутбуци, предлагат на потребителите гъвкавост и мобилност за изпълнение на разнообразни задачи - от основни приложения за производителност до софтуерни пакети с интензивни ресурси, без да са привързани към фиксирано място. Те разполагат с вградена клавиатура, дисплей и тракпад, входни и изходни портове. Освен това те са оборудвани с мощни процесори, памет и дисплеи с висока разделителна способност и батерии, осигуряващи продължителна мобилна употреба без необходимост от презареждане.



Фигура 1.21: HP лаптоп

Финален избор на дисплей и начин на управление

Въпреки компактния си форм-фактор и простотата си, 16x2 LCD Keypad Shield има сравнително малка площ на дисплея и ограничена разделителна способност, което ограничава количеството информация, което може да се показва, и не е подходящо за изискванията на проекта. От друга страна, 7,5-инчовият електронен хартиен дисплей със сензорно докосване, въпреки че осигурява по-голяма и визуално по-привлекателна площ на дисплея, страда от по-бавна честота на опресняване, което води до забележими забавяния при актуализиране на съдържанието на дисплея. Освен това сензорната функционалност на дисплеите e-Paper може да покаже пониска чувствителност или отзивчивост в сравнение с традиционните сензорни дисплеи, което може да повлияе на потребителското изживяване, особено при бързото темпо на работа, което изисква проектът. Ето защо беше избрана възможността за свързване на системата за търсене и проследяване с персонален компютър. Той има всички изисквания, необходими за показване на информация и управление на системата, и позволява да се намали обемът на изчисленията, които трябва да се извършват от микроконтролера, като позволява показването на информация да се извърши от процесора на компютъра. Освен това двете устройства могат много лесно да се свържат помежду си чрез серийна комуникация.

1.2.5 Източник на захранване

Литиево полимерни батерии

Литиево полимерните (Фиг. 1.22) батерии предлагат висока енергийна плътност и осигуряват продължителен период на работа, като същевременно поддържат компактен и лек физически размер, което ги прави идеални за приложения, при които ограниченията на пространството и теглото са от значение. Освен това литиевите батерии се отличават с висока скорост на разреждане, което им позволява да осигуряват достатъчно енергия за сложни задачи и внезапни скокове в нуждата от енергия, като например за управление на серво мотори. Освен това литиевите батерии се характеризират с широк температурен диапазон, което им позволява да функционират ефективно при различни условия на околната среда - от екстремно ниски до високи температури.



Фигура 1.22: Двуклеткова литиево полимерна батерия

Алкални батерии

Алкалните батерии (Фиг. 1.23) предлагат стабилно изходно напрежение и постоянна мощност през целия си живот, което гарантира предсказуема работа при различни условия на работа.



Фигура 1.23: Разнообразни алкални батерии

Освен това те се отличават с дълъг срок на годност и ниска степен на саморазреждане, което ги прави подходящи за приложения, изискващи периодична или рядка употреба. Въпреки че алкалните батерии могат да имат по-ниска енергийна плътност в сравнение с литиевите алтернативи, те компенсират това с по-ниска първоначална цена и лесна подмяна, което е привлекателно за проекти с бюджетни ограничения или когато подмяната на батериите е лесна. Освен това алкалните батерии се характеризират с добра работа в умерен температурен диапазон, което позволява използването им в широк спектър от вътрешни и външни среди. Характерът им за еднократна употреба елиминира необходимостта от сложна инфраструктура за зареждане или системи за управление на батериите, което опростява проектирането и внедряването на вградени проекти.

Финален избор на източник на захранване

За захранване на серводвигателите е избрана литиева батерия, тъй като тя има по-висока енергийна плътност, стабилно изходно напрежение и много по-висок ток на разреждане, който не може да бъде постигнат от други източници на захранване, като алкалните батерии или пинове на микроконтролер, нужен за контролиране на серво мотори. Тези характеристики осигуряват постоянно и ефективно подаване на енергия, което е от решаващо значение за прецизното управление и бързата реакция при работата на серводвигателите. Освен това литиевите батерии предлагат по-дълъг срок на годност и по-голяма устойчивост на температурни колебания, което повишава надеждността и дълготрайността на вградените системи. Въпреки че алкалните батерии са надеждни и рентабилни, превъзходните характеристики и ефективност на литиевите батерии ги правят предпочтитан избор за взискателни приложения като управлението на серводвигатели. Освен това за захранване на сензора и микроконтролера ще се използва съществуващата връзка USB между блока за дисплей и управление и блока за микроконтролер, тъй като тя осигурява необходимото напрежение и ток.

1.3 Използвани технологии, стандартни и протоколи за комуникация

1.3.1 Широчинно импулсна модулация (PWM)

Широчинно-импулсната модулация (ШИМ) е широко използвана техника в електрониката за управление на средната мощност, подавана към товара, чрез промяна на ширината на поредица от импулси с фиксирана амплитуда и честота. Обикновено се използва във вградени системи за задачи като управление на скоростта на двигатели, регулиране на интензитета на светодиоди и генериране на аналогови сигнали. ШИМ работи чрез бързо превключване на цифров сигнал между състояния ON (високо) и OFF (ниско), като съотношението на времето, прекарано във всяко състояние, определя средното напрежение или ток, подаван към товара. Чрез регулиране на работния цикъл, който представлява частта от времето, през което сигналът е в състояние ON (включено) в рамките на всеки период, може да се контролира точно ефективното изходно ниво. ШИМ предлага няколко предимства, включително ефективно подаване на енергия, простота на изпълнение с помощта на цифрови схеми и съвместимост с широк спектър от микроконтролери и интегрални схеми.

1.3.2 Универсална серийна шина (USB)

USB (Universal Serial Bus - универсална серийна шина) е широко разпространен стандарт за свързване на периферни устройства и устройства към компютри и други хост системи. Разработен в средата на 90-те години на миналия век от консорциум от технологични компании, USB се превърна в универсален интерфейс, предлагаш високоскоростен пренос на данни, захранване и функционалност "plug-and-play". USB конекторите са с различни форми и размери, като най-често срещаните са Type-A, Type-B и Type-C. USB поддържа множество скорости за пренос на данни, вариращи от оригиналната спецификация USB 1.1 със скорост до 12 Mbps до най-новата спецификация USB 3.2, която може да достигне скорост до 20 Gbps. USB също така предоставя възможности за доставка на енергия, което позволява на устройствата да черпят енергия от хост системата или да доставят енергия на свързаните периферни устройства. Освен това USB поддържа гореща подмяна, което позволява на потребителите да свързват и изключват устройства, без да изключват хост системата.

1.3.3 I2C

I2C (Inter-Integrated Circuit) е широко използван сериен комуникационен протокол, предназначен за ефективно свързване на множество интегрални схеми във вградени системи. Разработен от Philips Semiconductor (сега NXP Semiconductors) през 80-те години на миналия век, I2C улеснява комуникацията между главни и подчинени устройства по обща шина, като позволява едновременен двупосочен трансфер на данни. Тя работи с помощта на два проводника: серийна линия за данни (SDA) за

предаване на данни и сериен тактова линия (SCL) за синхронизиране на предаването на данни. I2C поддържа конфигурации с няколко главни устройства, което позволява на няколко устройства да комуникират по една и съща шина, и позволява гореща подмяна на устройствата, без да се прекъсва работата на шината. На всяко устройство на шината I2C е зададен уникален адрес, което улеснява безпроблемната комуникация и позволява на главните устройства да адресират конкретни подчинени устройства за обмен на данни.

1.3.4 [UART](#)

UART (универсален асинхронен приемник/предавател) е често използван комуникационен протокол, който позволява сериен предаване на данни между електронни устройства. Той служи като основен компонент в различни вградени системи, като улеснява обмена на данни между микроконтролери, сензори, изпълнителни механизми и други периферни устройства. UART работи асинхронно, което означава, че данните се предават без необходимост от общ часовников сигнал между изпращача и приемника. Вместо това той разчита на предварително определени скорости на предаване на данни, за да синхронизира предаването и приемането на данни. Комуникацията по UART включва два извода: един за предаване на данни (TX) и друг за приемане на данни (RX). Данните се предават последователно, като всеки байт обикновено се състои от стартови и стоп битове, както и от същинските битове данни.

1.4 САПР използвани в дипломния проект

1.4.1 FreeCAD

FreeCAD (Фиг. 1.24) е мощен параметричен 3D CAD (Computer-Aided Design) софтуер за моделиране с отворен код, предназначен за инженерни и дизайнърски проекти. Той предлага цялостен набор от инструменти, внимателно разработени за улесняване на прецизното и ефективно създаване, модифициране и анализ на сложни 3D модели. Със своя интуитивен потребителски интерфейс и богат набор от функции FreeCAD дава възможност на инженери, дизайнери и архитекти да реализират творческите си виждания с безпрецедентна гъвкавост и точност. Използвайки надеждните му възможности за параметрично моделиране, потребителите могат лесно да определят и манипулират геометрични ограничения, като гарантират целостта на проекта по време на итеративния процес. Освен това FreeCAD поддържа широк набор от файлови формати, което позволява безпроблемна съвместимост с друг CAD софтуер и улеснява съвместните работни процеси. Независимо дали става дума за концептуализиране на прототипи, симулиране на механични възли или генериране на подробни технически чертежи, FreeCAD е незаменим инструмент за професионалистите, които се стремят да превърнат идеите си в осезаема реалност с максимална прецизност и ефективност.



Open Source parametric 3D CAD modeler

Фигура 1.24: Лого на FreeCAD

1.4.2 KiCAD

KiCad (Фиг. 1.25) е широко използван софтуерен пакет с отворен код, предназначен за автоматизация на електронното проектиране, който служи като цялостен инструмент за заснемане на схеми, оформление на печатни платки и създаване на отпечатъци на компоненти. Неговият интуитивен потребителски интерфейс и обширна библиотека от компоненти позволяват на потребителите да работят с лекота. Освен това мощният редактор за оформление на печатни платки на KiCad предоставя усъвършенствани функции, като например правила за проектиране с възможност за персонализиране, автоматизирано маршрутизиране на следи и 3D визуализация, улеснявайки създаването на проекти на печатни платки от професионално ниво. Освен това KiCad поддържа безпроблемна интеграция с други инструменти на EDA чрез стандартизириани файлови формати, като осигурява съвместимост и оперативна съвместимост при различни работни процеси за проектиране.



Фигура 1.25: Лого на KiCAD

1.5 Програмна среда и програмни езици използвани в дипломния проект

1.5.1 Visual Studio Code

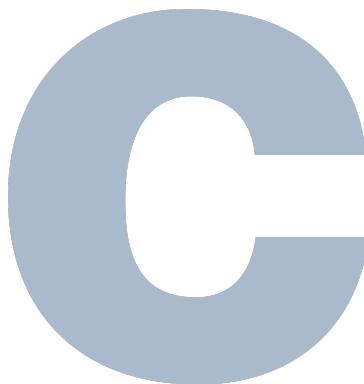
Visual Studio Code (Фиг. 1.26) е среда за разработване на код с отворен код, който поддържа много езици и функции. Той предлага интелигентен IntelliSense за автоматично довършване въз основа на типове променливи, дефиниции на функции и импортиирани модули, вградена Git интеграция за контрол на версии и възможности за отстраняване на грешки. Една от ключовите силни страни на VS Code е неговата разширяемост, позволяваща добавянето на нови езици, теми, програми за отстраняване на грешки и връзки към допълнителни услуги чрез разширения. За разработването на вграденото устройство в този проект е използвано разширението ESP-IDF. Разширението ESP-IDF е предназначено за разработване, изграждане, флаш, наблюдение и отстраняване на грешки в чипове Espressif с помощта на Espressif IoT Development Framework (ESP-IDF). Той осигурява безпроблемна интеграция с ESP-IDF, като предлага съветник за настройка за бърза инсталация на ESP-IDF и съответната верига от инструменти. Разширението поддържа широк набор от функции, включително GUI Menu Config за конфигурация на чип, и System View Tracing Viewer за потребителски интерфейс за проследяване.

The screenshot shows the Visual Studio Code interface. The code editor on the right displays Scheme code for a file named 'cafe.ss'. The code defines several procedures, including 'default-prompt-and-read', 'waiter-prompt-and-read', and 'waiter-write'. The code editor has syntax highlighting and a status bar at the bottom indicating 'Ln 34, Col 14'. The Explorer sidebar on the left shows the project structure, including files like 'a6fb.def', 'a6le.def', and various 'def' and 'ss' files. The 'SCHEME' folder contains many files starting with 'a' and 'c'. The bottom status bar also shows 'Spaces: 3', 'UTF-8', 'LF', and 'Scheme'.

Фигура 1.26: Код написан във Visual Studio Code

1.5.2 Програмен език С

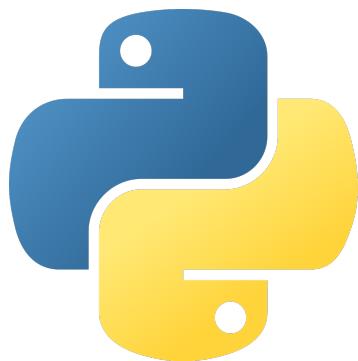
Езикът за програмиране С (Фиг. 1.27) е мощен и широко използван процедурен език за програмиране, първоначално разработен от Денис Ричи в началото на 70-те години на миналия век в Bell Labs. Известен със своята ефективност, гъвкавост и преносимост, С се превърна в крайъгълен камък в разработването на софтуер, особено за системно програмиране и вградени системи. Неговият синтаксис, повлиян от езика за програмиране В, набляга на простотата и гъвкавостта, което позволява на разработчиците да пишат кратък и ефективен код за широк спектър от приложения. С предлага достъп на ниско ниво до хардуерните компоненти и системните ресурси, което го прави подходящ за задачи, изискващи директна работа с паметта и прецизен контрол върху системните операции. Освен това неговите стандартизирана спецификации и широката поддръжка на библиотеки улесняват междуплатформената съвместимост и повторната използваемост на кода в различни среди. Въпреки че е по-стар език, С продължава да бъде актуален и широко разпространен в различни области, включително операционни системи, компилатори, вградени системи и високопроизводителни изчисления, което показва неговото трайно значение в областта на езиците за програмиране.



Фигура 1.27: Лого от корицата на първото издание на програмния език С

1.5.3 Програмен език Python

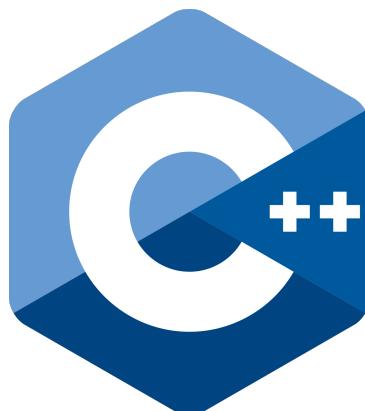
Python (Фиг. 1.28) е интерпретиран език за програмиране от високо ниво, известен със своята простота, разбираемост и гъвкавост. Създаден от Гуидо ван Росум и пуснат за първи път през 1991 г., оттогава Python е широко разпространен в различни области, включително уеб разработване, наука за данните, изкуствен интелект, научни изчисления и др. Неговият изчистен и кратък синтаксис, включващ блоково структуриране, базирано на отстъпите, го прави лесен за изучаване и разбиране, дори за начинаещи. Обширната стандартна библиотека на Python предлага богат набор от модули и функции за изпълнение на широк спектър от задачи - от обработка на файлове и работа в мрежа до математически изчисления и достъп до бази данни, което намалява необходимостта от външни зависимости. Освен това динамичното типизиране и автоматичното управление на паметта на Python допринасят за неговата гъвкавост и лекота на използване, като позволяват на разработчиците да се съсредоточат върху решаването на проблеми, а не върху управлението на детайли от ниско ниво.



Фигура 1.28: Лого на програмният език Python

1.5.4 Програмен език C++

C++ (Фиг. 1.29) е мощен, обектно-ориентиран език за програмиране, известен със своята производителност, гъвкавост и широка употреба в различни области. Възникнал като разширение на езика за програмиране C, C++ е разработен от Барне Строуструп в началото на 80-те години на миналия век, за да предостави допълнителни функции, като класове, наследяване, полиморфизъм и шаблони, позволяващи по-ефективно и организирано разработване на софтуер. C++ предлага богат набор от библиотеки и рамки, което го прави подходящ за създаване на разнообразни приложения, вариращи от системен софтуер, игри и графични потребителски интерфейси до вградени системи, приложения в реално време и високопроизводителни изчисления. Силното му статично типизиране и проверката по време на компилиация допринасят за стабилността и надеждността, а поддръжката на процедурни и обектно-ориентирани парадигми на програмиране подобрява многократната използваемост и поддържането на кода. Освен това C++ осигурява контрол на ниско ниво върху хардуерните ресурси и управлението на паметта, като улеснява разработването на ефективен и оптимизиран код за приложения с критично значение за производителността.



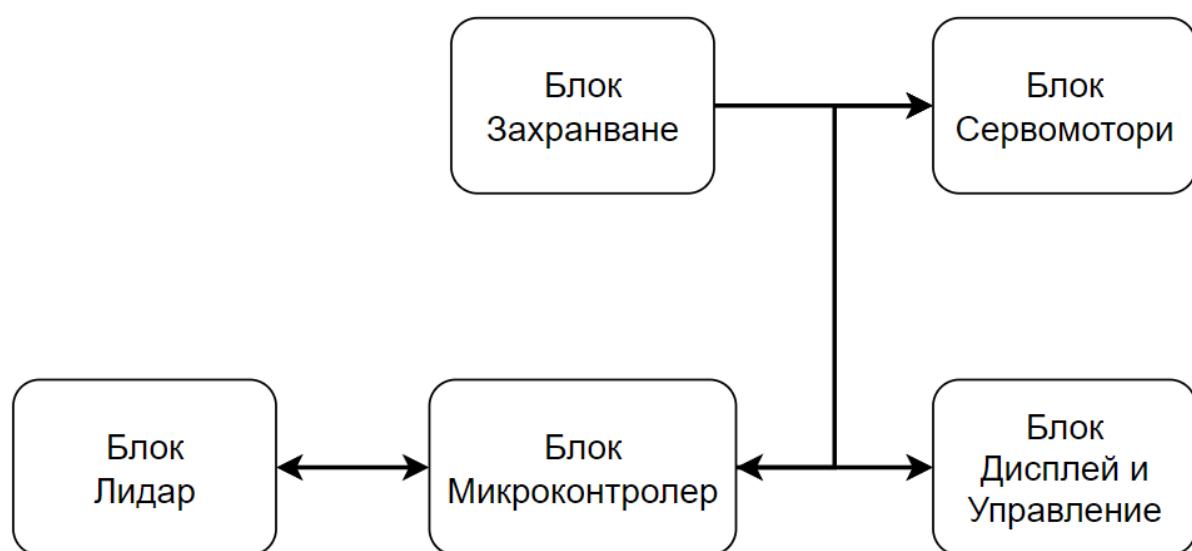
Фигура 1.29: Лого одобрено от комитета за стандарти C++

ВТОРА ГЛАВА

2.1 Основни изисквания към дипломния проект

Целта на проекта е да се изгради устройство на базата на микроконтролер ESP32, чрез който да може да се сканира избрана порция от небето, да се извеждат данни за няколко цели и да има възможност една от тях да бъде избрана и проследявана, като се изчислява и представя информация за разстояние, височина, вектор на движение и скорост. Устройството трябва да позволява изчисляване на огнево решение за избраната цел. Корпусът на устройството трябва да предпазва компонентите от прах и опръскване с вода.

2.2 Блокова схема на устройството



2.2.1 Описание на блоковата схема на устройството

Блок Микроконтролер

Блокът микроконтролер служи като централен процесор на вградената система, отговорен за изпълнението на програмната логика и взаимодействието с периферните устройства. Той се състои от ESP32 DEVKIT V1 микроконтролерът, избран за дипломният проект. Блокът е захранван от устройството в блок дисплей и управление и комуникира с него чрез UART. Блокът също така комуникира чрез ШИМ с блок серво мотори, и захранва и комуникира чрез I2C с блок лидарен сензор.

Блок Захранване

Блокът за захранване се състои от двуклеткова литиево полимерна батерия, която захранва единствено блок серво мотори.

Блок Сервомотори

Блокът за сервоздвижвания включва два MG996R сервомотори, които са управлявани чрез ШИМ от блок микроконтролер. Те отговарят за прецизното насочване на блок лидарен сензор.

Блок Лидар

В този блок е използван лидарният сензор Garmin LIDAR-Lite v4, като може да бъде заменен с по-способната трета версия на този сензор, без нуждата от друга промяна в устройството. Той отговаря за изпращане на измерено разстояние към обекти пред него към блок микроконтролер.

Блок Дисплей и Управление

Блок дисплей и управление служи за да задават получава и изпраща команди към системата и да визуализира калкулираната информация в удобен вид. За целта е използван преносим компютър, който чрез USB връзка захранва и комуникира с блок микроконтролер.

ТРЕТА ГЛАВА

3.1 Проектиране на принципна електрическа схема на системата за търсене и проследяване на цели

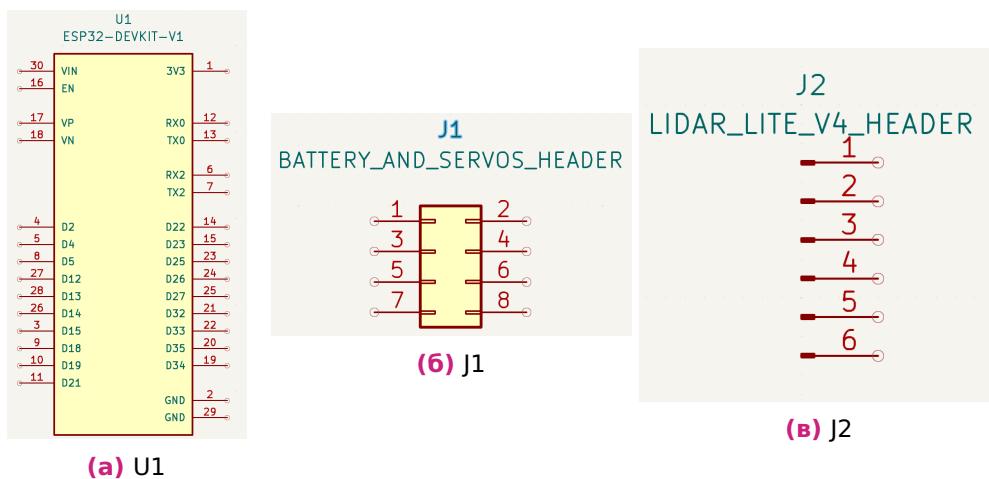
3.1.1 Избор на компоненти

Микроконтролер

В принципната електрическа схема, използваният микроконтролер (U1) е ESP32 DEVKIT V1, чийто символ е импортиран от сайт за символи и графични изображения на корпуси на елементи.

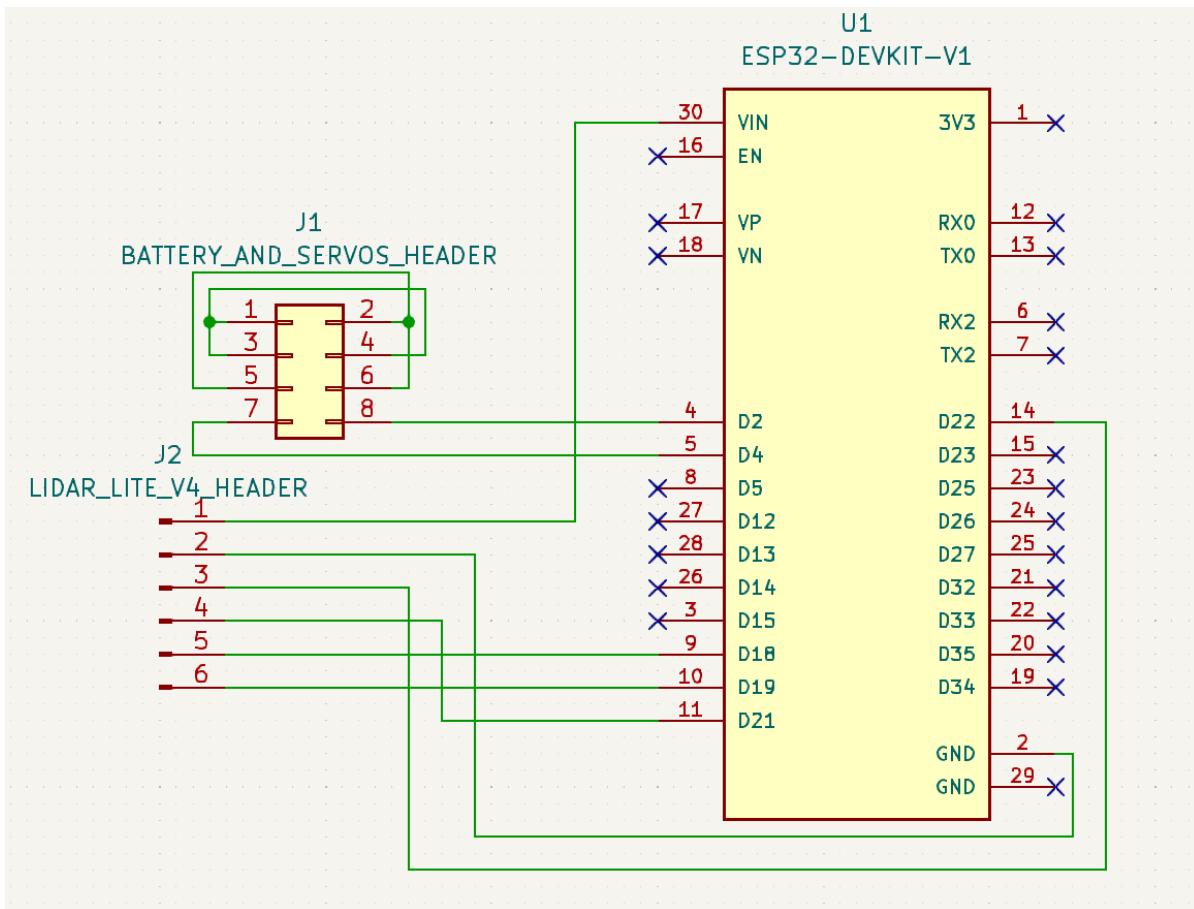
Конектори

Първият използван конектор (J1) с осем, разделени на два реда, мъжки пина е предназначен за свързване на серво моторите със микроконтролера и захранването от батерията. Вторият използван конектор (J2) с шест мъжки пина е предназначен за свързване на лидарния сензор с микроконтролера. Тези конектори позволяват лесно и модулено свързване на компонентите в системата.



Фигура 3.1: Символи на компоненти в KiCAD. (а) Тук е показан символа на ESP32 Devkit V1.(б) Тук е показан символа на конектора използван за свързване на сервомоторите, батерията и микроконтролера. (б) Тук е показан символа на конектора използван за свързване на лидарния сензор с микроконтролера.

3.1.2 Принципна електрическа схема



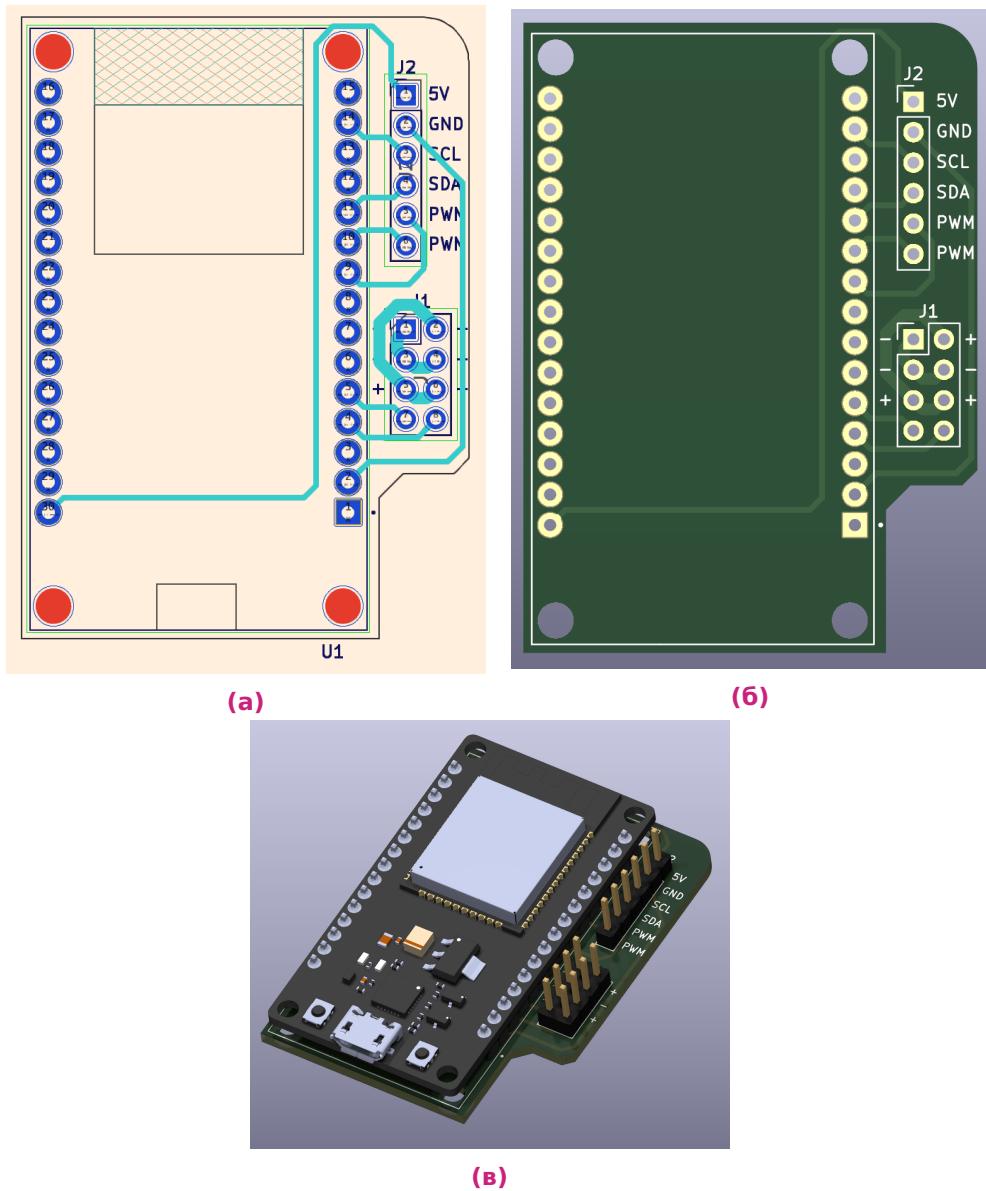
Фигура 3.2: Принципна електрическа схема на устройството

Във фиг. 3.2 е показана принципната електрическа схема на устройството. Пиновете D2 и D4 на микроконтролерът (U1) са свързани към J1, като през тях се управляват сервомоторите чрез ШИМ сигнали. Неговите SCL и SDA пинове (D21 и D22), GND, VIN, D18 и D19 са свързани с конектора за лидарният сензор, като му предоставят захранване и комуникация чрез I2C. Първи и втори пин на J1 се използват за връзка с батерията и са свързани с пиновете предназначени за двета сервомотора, за да предоставят захранване.

ЧЕТВЪРТА ГЛАВА

4.1 Информация за печатната платка

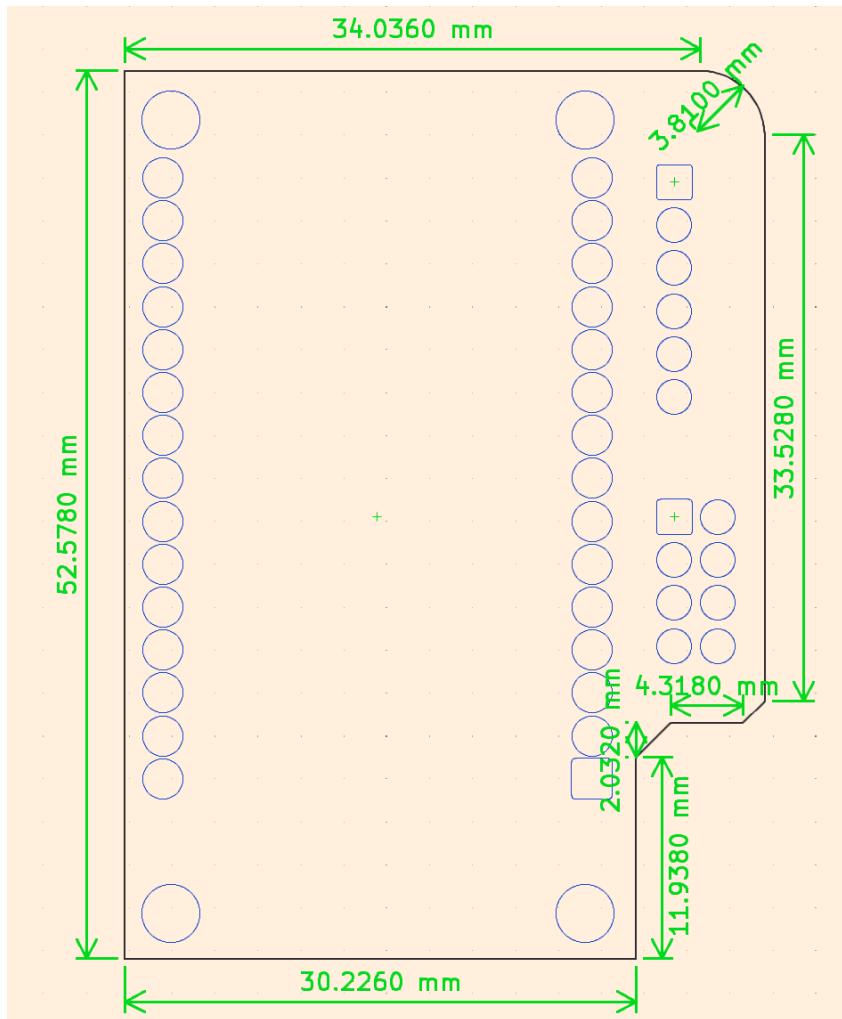
Печатната платка е еднослойно опроводена, като има 4 отвора за монтаж върху главния корпус на устройството. Тя е показана на фиг. 4.1.



Фигура 4.1: Платката е показвана в KiCAD. (а) Тук е показвана платката в редактора на печатни платки на KiCAD. (б) Тук е показвана платката без компоненти с опцията за триизмерен изглед на KiCAD. (б) Тук е показвана платката с компоненти с опцията за триизмерен изглед и проследяване на лъчи на KiCAD.

Размер

Печатната платка има сложна форма с площ от 1881.5 mm², като нейните размери са погазани на фиг. 4.2



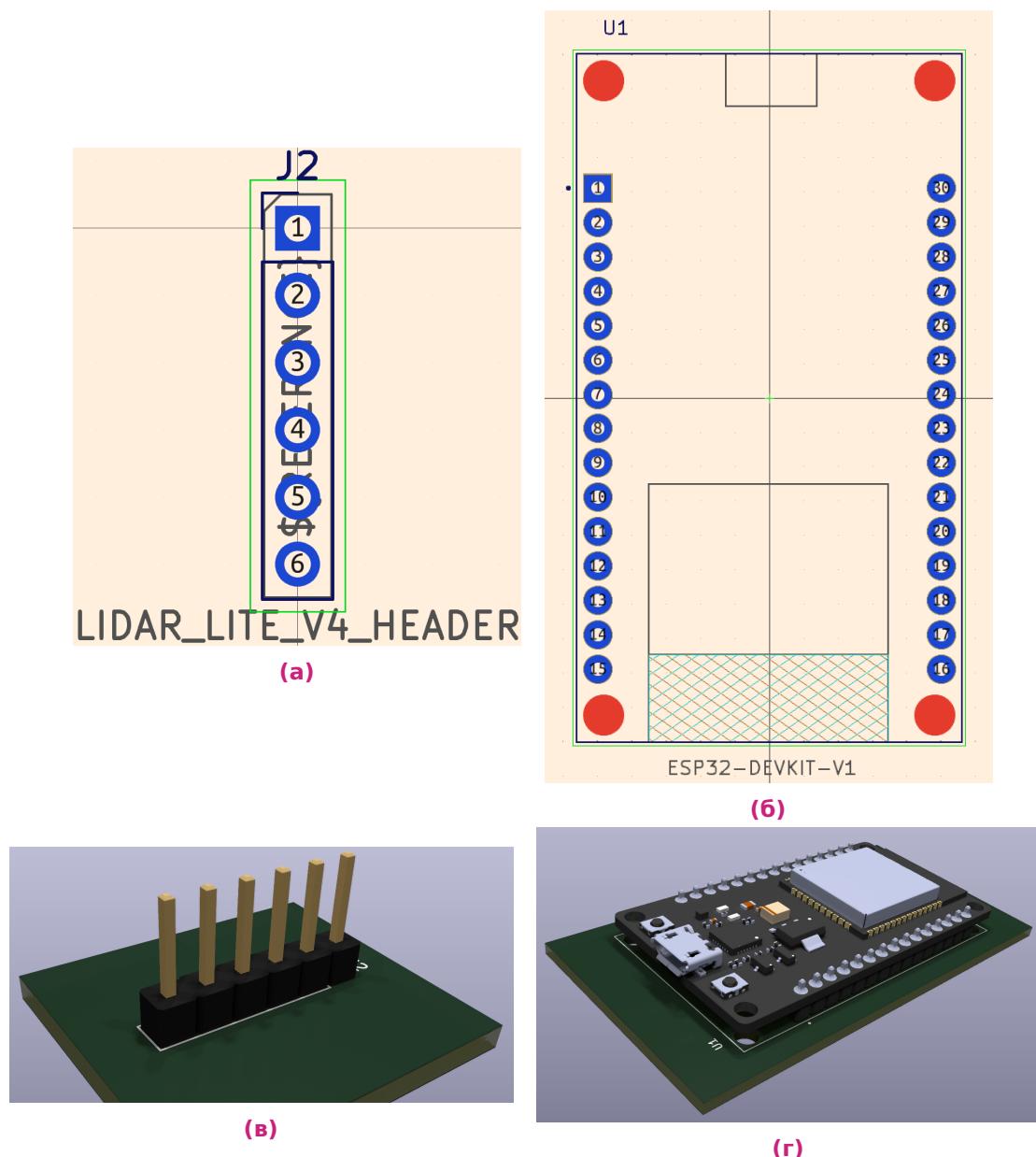
Фигура 4.2: Размери на печатната платка

Писти

Всички писти са направени с ширина половин милиметър, поради лимитации от производителя, като пистите за захранване на двата серво мотора са направени с ширина един милиметър поради по-големия ток проптичащ през тях.

4.2 Проектиране на печатна платка за системата за търсене и проследяване на цели

4.2.1 Графични изображения на корпуси на елементи и триизмерни модели на използваните компоненти



Фигура 4.3: На фигурата са показани следните графични изображения на корпуси на елементи и триизмерни модели на използваните компоненти: (а) и (в) показват един от конекторите. (б) и (г) показват микроконтролера.

Микроконтролер

За графично изображение на корпуса и триизмерен модел на използваният микроконтролер (U1) е ESP32 DEVKIT V1, е използван сайтът snapeda.com, от който те са импортирани в програмата.

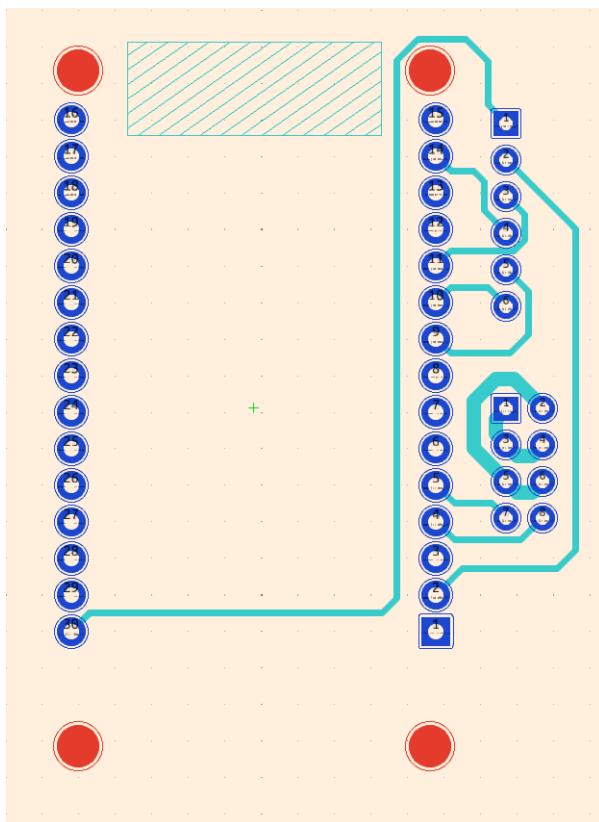
Конектори

Първият използван конектор (J1) използва корпус с осем, разделени на два реда, мъжки пина и е показан на фиг. Вторият използван конектор (J2) с шест мъжки пина в един ред и е показан на фиг. За тях са използвани корпуси и триизмерни модели от библиотеките на KiCAD.

4.2.2 Слоеве на печатната платка

Преден и заден меден слой (F.Cu и B.Cu)

Предният меден слой представлява опроводящите писти и подложки от горната страна на печатната платка, които улесняват електрическите връзки между компонентите. Задният слой, изпълнява същата функция от задната страна на платката, но тъй като тя е еднослойна, той не се използва.



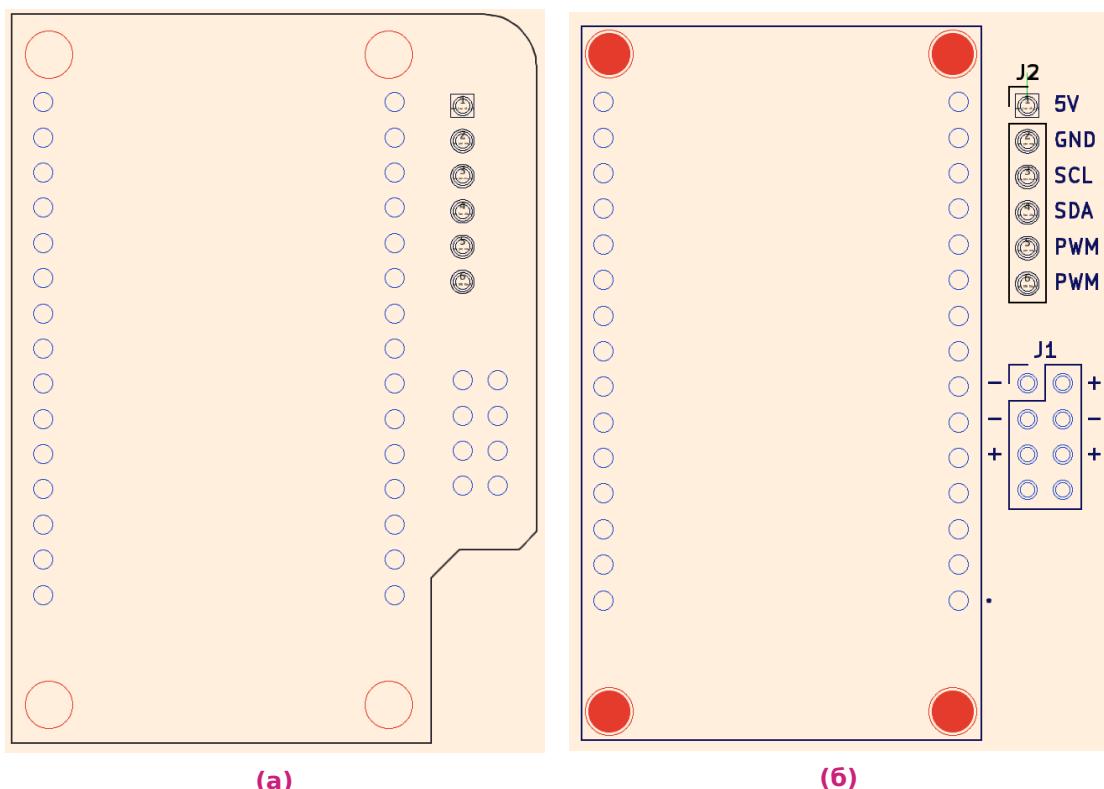
Фигура 4.4: Преден меден слой

Слой изрязване на ръбове (Edge Cuts)

Слоят Edge Cuts определя контура на печатната платка, очертавайки нейната форма и размери и служейки като ръководство за процесите на изработка и монтаж.

Преден и заден ситопечат (F.SilkS и B.SilkS)

Тези слоеве включват маркировки със ситопечат, като например очертания на компоненти, етикети и символи, поставени върху горната страна на печатната платка за идентифициране на компонентите и насочване на монтажа. Използван само предният слой, като на него са означени за какво служат различните пинове на конекторите, за лесно свързване с модулите.



Фигура 4.5: Слой изрязване на ръбове (а) и преден ситопечат (б).

Предна и задна защитна маска (F.Mask)

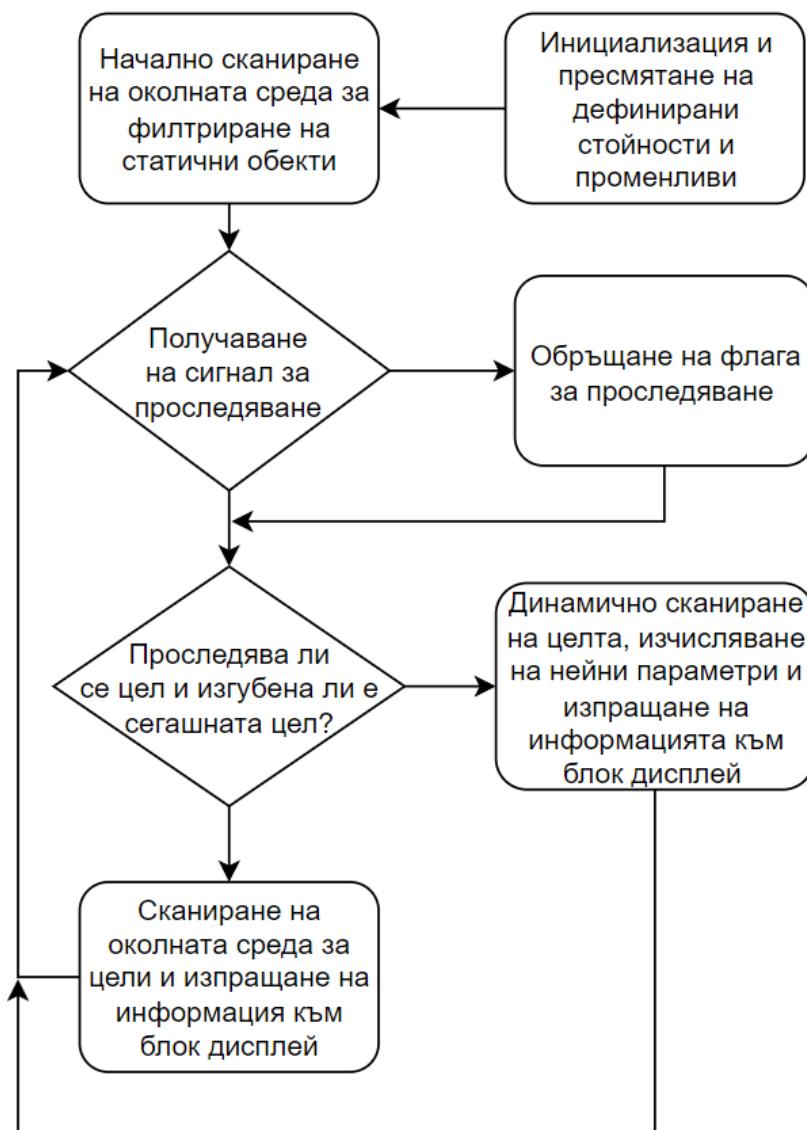
Тези слоеве определя областите от горната страна на печатната платка, където се нанася маска за запояване, предпазваща медните следи от окисляване и къси съединения по време на запояването.

Монтажни отвори (Drill Holes)

Този слой определя местата и размерите на пробитите отвори на печатната платка, включително отворите за монтиране на компоненти, проходните канали и други елементи, изискващи пробиване през платката. Четирите отвора на този слой служат за монтаж върху главния корпус на устройството.

ПЕТА ГЛАВА

5.1 Блокова схема на управляващ софтуер за системата за търсене и проследяване на цели



Фигура 5.1: Блокова схема на кода

5.2 Управляващ софтуер за системата за търсене и проследяване на цели

```
1 #include <stdio.h>
2 #include <math.h>
3 #include <driver/gpio.h>
4 #include "freertos/FreeRTOS.h"
5 #include "freertos/task.h"
6 #include "esp_system.h"
7 #include "esp_timer.h"
8 #include "driver/mcpwm.h"
9 #include "driver/uart.h"
10 #include <cctype.h>
11 #include "soc/mcpwm_periph.h"
12 #include "LIDARLite_v4LED.h"
13
14 LIDARLite_v4LED myLIDAR;
15
16 // Define servo pins (Beta is the vertically turning servo motor) and PWM channels
17 #define SERVO_PIN_ALPHA 3
18 #define SERVO_PIN_BETA 5
19 #define SERVO_MIN_PULSEWIDTH 500 // Minimum pulse width in microseconds
20 #define SERVO_MAX_PULSEWIDTH 2500 // Maximum pulse width in microseconds
21 #define SERVO_MAX_DEGREE 180
22 // Define lidar measurements per second and time to be simulated
23 #define tick_rate 15
24 float cycle_duration_ms = 1000/tick_rate;
25
26 // Define max lidar detection range for very large target in meters, divergence in
27 // degrees and area to be scanned in degrees
28 // Set vertical scan to the desired furthest up, Set horizontal scan to the
29 // desired horizontal degrees
30 #define max_lidar_range 250
31 #define lidar_divergence 5
32 #define vertical_degrees_scanned 20
33 #define horizontal_degrees_scanned 60
34
35 // Define the times acquiring needs to not find a target before acquisition is
36 // canceled and the size of the starting acquisition box is in degrees (The box
37 // changes size dynamically based on target size)
38 #define acq_width 10
39 #define acq_height 6
40 #define ttl_scans 5
41
42 // Variables used for keeping track of the TTL of the acquisition and the dynamic
43 // acquisition box
44 int high_acq_edge, low_acq_edge, left_acq_edge, right_acq_edge;
45 int ttl;
46
47 // Arrays to store X, Y, Z coordinates of the target at two different points in
48 // time, used for data calculation
49 float first_target_coordinates[3];
```

```

44 float second_target_coordinates[3];
45
46 // Variables for target data
47 float target_speed, old_target_speed, acceleration, heading;
48
49 // First bool is used for alternating between which X, Y, Z coordinate array is
50 // being written to, second represents whether the system is tracking a target
51 bool first_scan = true;
52 bool acquiring = false;
53
54 // Calculate largest area that can be scanned without leaving areas unscanned with
55 // the formula "(lidar_divergence / 2) * sqrt(2)"
56 float bar_size = 3.5;
57 float vertical_bar_size = 3.5;
58
59 // Define values used for servo guidance in degrees (-90 to +90 for horizontal, 0
60 // to +90 for vertical)
61 float alpha_servo_degrees = 0, beta_servo_degrees = 0;
62 int alpha_servo_bars = 0, beta_servo_bars = 0;
63
64 // 2D array for storing distances to ground clutter for filtration
65 // IMPORTANT: SIZE MUST BE PRECALCULATED BASED ON DIVERGENCE SO THAT IT ISN'T TOO
66 // SMALL
67 float ground_clutter[36][18];
68
69 // Sensor's maximum error in meters. Used to prevent returning clutter as a target
70 // due to imprecision in separate measurements.
71 float deviation = 0.5;
72
73 // Total cycles made
74 int current_cycle = 0;
75 int64_t nextCycleMillis;
76
77 size_t buffered_data_length;
78
79 // Bool used to keep track of vertical servo motor's rotation direction
80 bool seeking_up = false;
81
82 void init_pwm() {
83     mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0A, SERVO_PIN_ALPHA);
84     mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM1A, SERVO_PIN_BETA);
85     mcpwm_config_t pwm_config;
86     pwm_config.frequency = 50;
87     pwm_config.cmpr_a = 0;
88     pwm_config.cmpr_b = 0;
89     pwm_config.counter_mode = MCPWM_UP_COUNTER;
90     pwm_config.duty_mode = MCPWM_DUTY_MODE_0;
91     mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_0, &pwm_config);
92     mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_1, &pwm_config);
93 }
94
95 void set_servos_degrees(uint32_t angle_servo1, uint32_t angle_servo2) {
96     uint32_t duty_servo1, duty_servo2;
97     //Convert angle to duty cycle for servos

```

```

93     duty_servo1 = (angle_servo1 / (float)SERVO_MAX_DEGREE) * (SERVO_MAX_PULSEWIDTH
94         - SERVO_MIN_PULSEWIDTH) + SERVO_MIN_PULSEWIDTH;
95     duty_servo2 = (angle_servo2 / (float)SERVO_MAX_DEGREE) * (SERVO_MAX_PULSEWIDTH
96         - SERVO_MIN_PULSEWIDTH) + SERVO_MIN_PULSEWIDTH;
97     mcpwm_set_duty_in_us(MCPWM_UNIT_0, MCPWM_TIMER_0, MCPWM_OPR_A, duty_servo1);
98     mcpwm_set_duty_in_us(MCPWM_UNIT_0, MCPWM_TIMER_1, MCPWM_OPR_A, duty_servo2);
99 }
100
101 // Function to calculate the next cycle's millis
102 int64_t calculateNextCycleMillis() {
103     nextCycleMillis = cycle_duration_ms * current_cycle;
104     return nextCycleMillis;
105 }
106
107 // Void to move servos across search area without leaving spots unscanned one
108 // movement at a time.
109 void moveServos(){
110     set_servos_degrees(alpha_servo_degrees, beta_servo_degrees);
111     alpha_servo_bars = round(alpha_servo_degrees / fabs(bar_size));
112     beta_servo_bars = round(beta_servo_degrees / fabs(bar_size));
113
114     alpha_servo_degrees += bar_size;
115
116     if (alpha_servo_degrees >= horizontal_degrees_scanned) {
117         bar_size = -bar_size;
118         beta_servo_degrees += vertical_bar_size;
119     } else if (alpha_servo_degrees <= 0) {
120         bar_size = fabs(bar_size);
121         beta_servo_degrees += vertical_bar_size;
122     }
123     if (beta_servo_degrees >= vertical_degrees_scanned) {
124         vertical_bar_size = -fabs(vertical_bar_size);
125         if (seeking_up){
126             printf("\n");
127             seeking_up = !seeking_up;
128         }
129     } else if (beta_servo_degrees <= 0) {
130         if (!seeking_up){
131             printf("\n");
132             seeking_up = !seeking_up;
133         }
134     }
135 }
136
137 float lidarMeasurement(){
138     float newDistance;
139     newDistance = myLIDAR.getDistance();
140     newDistance = newDistance / 100;
141     return newDistance;
142 }
143

```

```

144 float calculateHeight(float distance) {
145     // Convert angles from degrees to radians
146     float verticalAngleRad = beta_servo_degrees * (M_PI / 180);
147
148     // Calculate the height (Z-coordinate) of the point
149     float height = distance * tan(verticalAngleRad);
150
151     return height;
152 }
153
154 int parseIntFromSerial() {
155     char buffer[16];
156     int index = 0;
157
158     // Read characters from serial until a non-numeric character is encountered
159     while (1) {
160         if (index < sizeof(buffer) - 1) {
161             int bytes_read = uart_read_bytes(0, (uint8_t *)&buffer[index], 1,
162                                             portMAX_DELAY);
163             if (bytes_read == 1) {
164                 if (isdigit((int)buffer[index])) {
165                     index++;
166                 } else {
167                     break;
168                 }
169             } else {
170                 break;
171             }
172         }
173
174         buffer[index] = '\0';
175
176         int parsed_int = atoi(buffer);
177
178         return parsed_int;
179     }
180
181     uint8_t readByteFromSerial() {
182         uint8_t byte;
183
184         int bytes_read = uart_read_bytes(0, &byte, 1, portMAX_DELAY);
185
186         if (bytes_read == 1) {
187             return byte;
188         } else {
189             return 0;
190         }
191     }
192
193
194     void targetAcquisition(){
195         int high_target_edge = 0, low_target_edge = 0, left_target_edge = left_acq_edge,
196             right_target_edge = 0;

```

```

196  for (int y_degrees = low_acq_edge; y_degrees <= high_acq_edge; y_degrees +=  

197      bar_size){  

198      for (int x_degrees = left_acq_edge; x_degrees <= right_acq_edge; x_degrees +=  

199          bar_size){  

200          set_servos_degrees(x_degrees, y_degrees);  

201          float lidar_distance_to_target = lidarMeasurement();  

202          if (lidar_distance_to_target < max_lidar_range){  

203              ttl = ttl_scans;  

204              low_target_edge = y_degrees;  

205              if(right_target_edge < x_degrees){  

206                  right_target_edge = x_degrees;  

207              }  

208              if(high_target_edge < y_degrees){  

209                  high_target_edge = y_degrees;  

210              }  

211              if(left_target_edge > x_degrees){  

212                  high_target_edge = x_degrees;  

213              }  

214              if (first_scan){  

215                  first_scan = !first_scan;  

216                  first_target_coordinates[0] = lidar_distance_to_target * cos(x_degrees *  

217                      (M_PI / 180)) * sin(y_degrees * (M_PI / 180));  

218                  first_target_coordinates[1] = calculateHeight(lidar_distance_to_target);  

219                  first_target_coordinates[2] = lidar_distance_to_target * sin(x_degrees *  

220                      (M_PI / 180)) * sin(y_degrees * (M_PI / 180));  

221              }  

222          else{  

223              first_scan = !first_scan;  

224              second_target_coordinates[0] = lidar_distance_to_target * cos(x_degrees *  

225                  (M_PI / 180)) * sin(y_degrees * (M_PI / 180));  

226              second_target_coordinates[1] = calculateHeight(lidar_distance_to_target);  

227              second_target_coordinates[2] = lidar_distance_to_target * sin(x_degrees *  

228                  (M_PI / 180)) * sin(y_degrees * (M_PI / 180));  

229          }  

230  

231          old_target_speed = target_speed;  

232          target_speed = sqrt(pow(second_target_coordinates[0] -  

233              first_target_coordinates[0], 2) + pow(second_target_coordinates[1] -  

234              first_target_coordinates[1], 2) + pow(second_target_coordinates[2] -  

235              first_target_coordinates[2], 2)) / cycle_duration_ms;  

236  

237          acceleration = target_speed - old_target_speed;  

238  

239          heading = atan((second_target_coordinates[2] - first_target_coordinates[2]) /  

240              (second_target_coordinates[0] - first_target_coordinates[0]));  

241          if ((second_target_coordinates[2] - first_target_coordinates[2]) < 0 &&  

242              (second_target_coordinates[0] - first_target_coordinates[0]) >= 0) {  

243              heading += M_PI;  

244          } else if ((second_target_coordinates[2] - first_target_coordinates[2]) < 0  

245              && (second_target_coordinates[0] - first_target_coordinates[0]) < 0) {  

246              heading -= M_PI;  

247          }

```

```

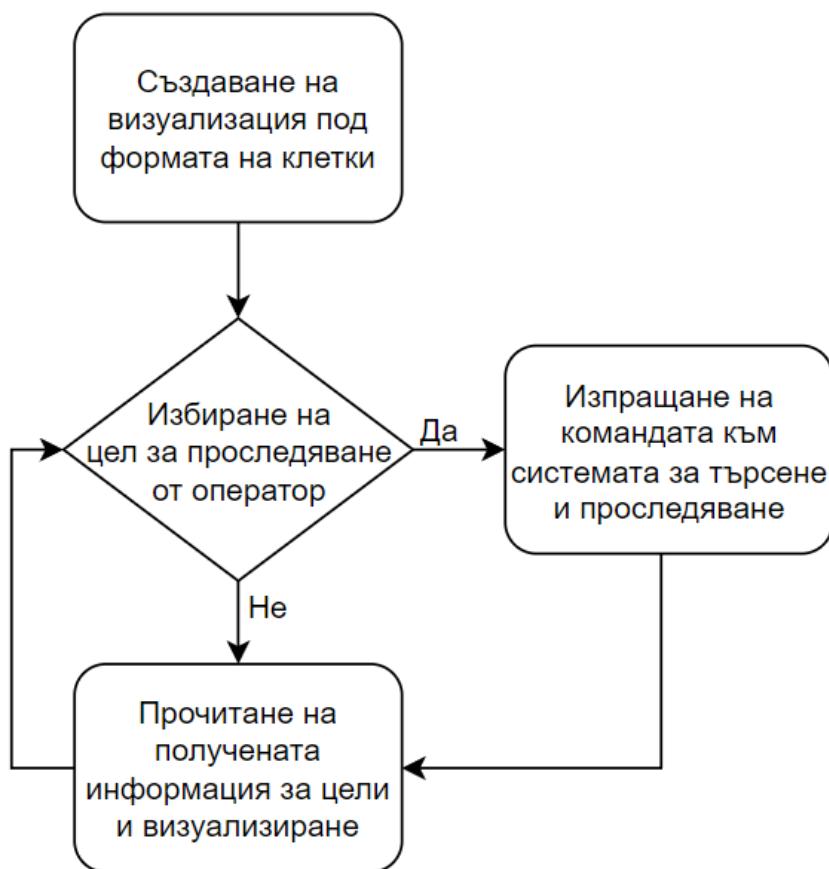
238     printf("Speed: %f Height: %f Acceleration: %f Heading: %f\n", target_speed,
239         calculateHeight(lidar_distance_to_target), acceleration, heading * (180 /
240             M_PI));
240 }
241
242 high_acq_edge = high_target_edge + bar_size;
243 low_acq_edge = low_target_edge - bar_size;
244 left_acq_edge = left_target_edge - bar_size;
245 right_acq_edge = right_target_edge + bar_size;
246 }
247
248
249 void app_main() {
250     printf("Initializing...\n");
251     init_pwm();
252     gpio_set_direction(SERVO_PIN_ALPHA, GPIO_MODE_OUTPUT);
253     gpio_set_direction(SERVO_PIN_BETA, GPIO_MODE_OUTPUT);
254
255     if (!myLIDAR.begin()) {
256         printf("Device did not acknowledge! Freezing.\n");
257         while(1);
258     }
259     printf("LIDAR acknowledged!\n");
260
261 // Initial scan for ground clutter filtration
262 while (beta_servo_degrees <= vertical_degrees_scanned || alpha_servo_degrees <=
263     horizontal_degrees_scanned){
264     moveServos();
265     ground_clutter[alpha_servo_bars][beta_servo_bars] = lidarMeasurement();
266     vTaskDelay(cycle_duration_ms / portTICK_PERIOD_MS);
267 }
268 while(1) {
269     // Wait until it's time to start the cycle
270     while (esp_timer_get_time() / 1000 <= calculateNextCycleMillis()) {}
271
272     if (uart_get_buffered_data_len(0, &buffered_data_length) >= 5) {
273         if (!acquiring){
274             ttl = ttl_scans;
275             acquiring = !acquiring;
276             low_acq_edge = parseIntFromSerial() - (acq_height/2);
277             readByteFromSerial();
278             left_acq_edge = parseIntFromSerial() - (acq_width/2);
279             readByteFromSerial();
280             right_acq_edge = left_acq_edge + acq_width;
281             high_acq_edge = low_acq_edge + acq_height;
282             } else {acquiring = !acquiring;}
283
284         if (ttl > 0 && acquiring){
285             ttl--;
286             targetAcquisition();
287         }else {
288             moveServos();

```

```
289     float lidar_distance_to_target;
290     lidar_distance_to_target = lidarMeasurement();
291     if(ground_clutter[alpha_servo_bars][beta_servo_bars] <=
292         lidar_distance_to_target + deviation){
293         lidar_distance_to_target = max_lidar_range;
294     }
295     if (lidar_distance_to_target < max_lidar_range){
296         printf("%f,", lidar_distance_to_target);
297     }else {printf("0");}
298
299     current_cycle += 1;
300 }
301 }
```

Програмен код 5.1: Пълен управляващ софтуер за микроконтролера.

5.3 Блокова схема на управляващ софтуер за визуализиране на информация и управляване на системата



Фигура 5.2: Блокова схема на кода

5.4 Управляващ софтуер за визуализиране на информация и управляване на системата

```
1 import serial
2 import numpy as np
3 import pygame
4
5
6 ser = serial.Serial('COM3', 115200)
7
8 pygame.init()
9
10 GRID_SIZE = 20
11 SCREEN_SIZE = 1200
12 GRID_WIDTH = SCREEN_SIZE // GRID_SIZE
13 WHITE = (255, 255, 255)
14 BLACK = (0, 0, 0)
15
16
17 def receive_data():
18     data = ser.readline().decode().strip()
19     if data and not data[0].isdigit():
20         print(data)
21
22     if data:
23         values = [int(val) for val in data.split(',') if val.isdigit()]
24         if len(values) == GRID_SIZE * GRID_SIZE:
25             array2D = np.array(values).reshape((GRID_SIZE, GRID_SIZE))
26             return array2D
27
28     return np.zeros((GRID_SIZE, GRID_SIZE), dtype=int)
29
30 def draw_grid(array2D):
31     screen.fill(BLACK)
32     for y in range(GRID_SIZE):
33         for x in range(GRID_SIZE):
34             if array2D[y][x] > 0:
35                 color = WHITE
36                 distance = 10 - array2D[y][x]
37                 pygame.draw.rect(screen, color, (x*GRID_WIDTH, distance*GRID_WIDTH,
38                                         GRID_WIDTH, GRID_WIDTH))
39
40     pygame.display.flip()
41
42 screen = pygame.display.set_mode((SCREEN_SIZE, SCREEN_SIZE))
43 clock = pygame.time.Clock()
44 running = True
45 array2D = np.zeros((GRID_SIZE, GRID_SIZE), dtype=int)
46
47 while running:
48     for event in pygame.event.get():
49         if event.type == pygame.QUIT:
50             running = False
51         elif event.type == pygame.MOUSEBUTTONDOWN:
```

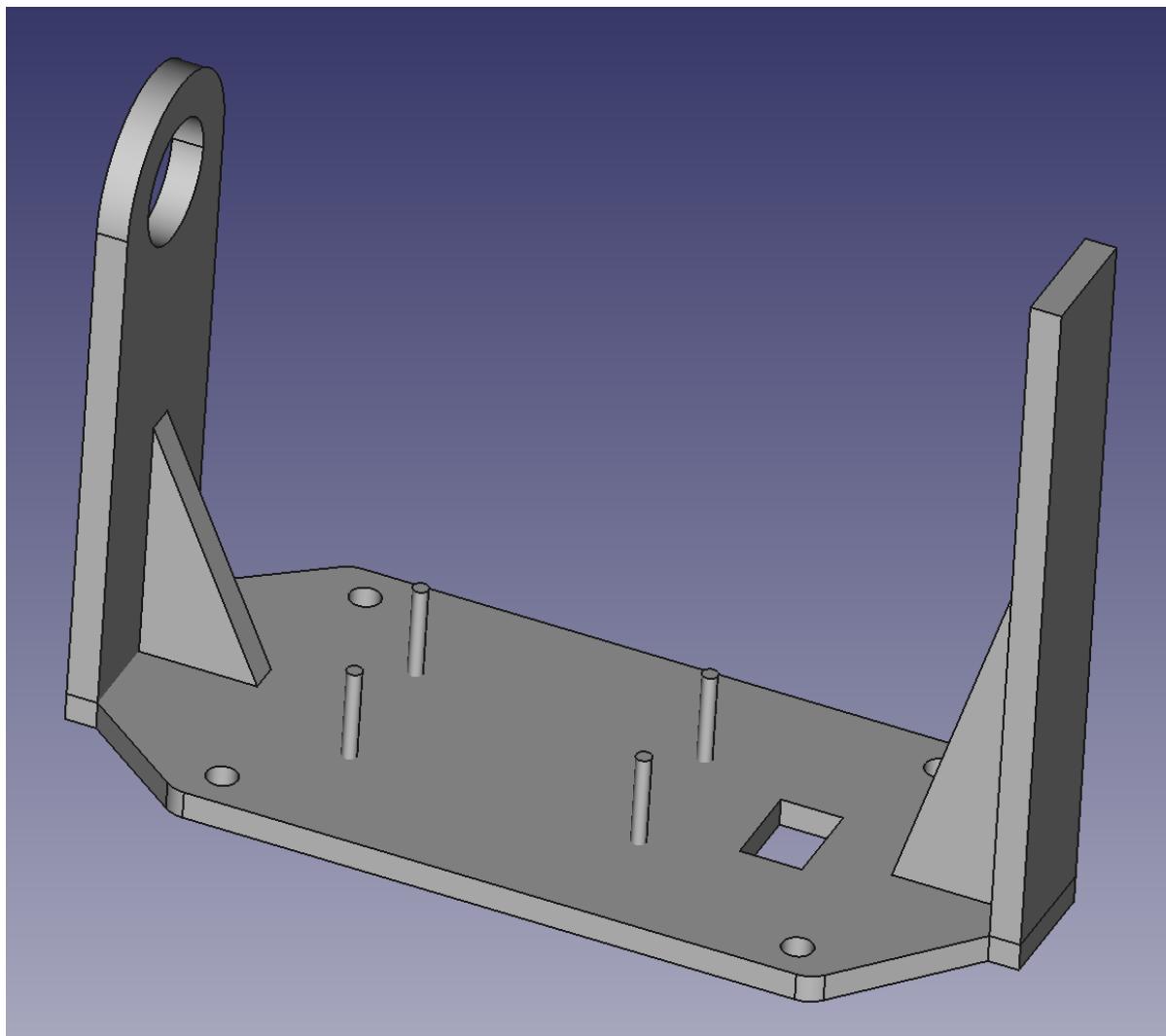
```
49         x, y = pygame.mouse.get_pos()
50         x_index = x // GRID_WIDTH
51         y_index = y // GRID_WIDTH
52
53     for i in range(20):
54         if array2D[i][x_index] == 10 - y_index:
55             ser.write(bytes(f"{i},{x_index}\n", 'utf-8'))
56
57     else:
58         pass
59
60 array2D = receive_data()
61 draw_grid(array2D)
62 clock.tick(30)
63
64 pygame.quit()
```

Програмен код 5.2: Пълен управляващ софтуер за визуализация на информация и комуникация с микроконтролер.

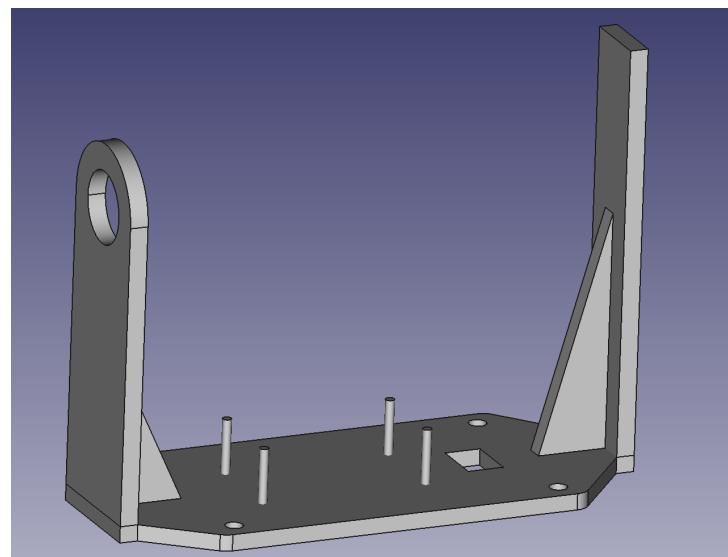
ШЕСТА ГЛАВА

6.1 Корпус на устройството

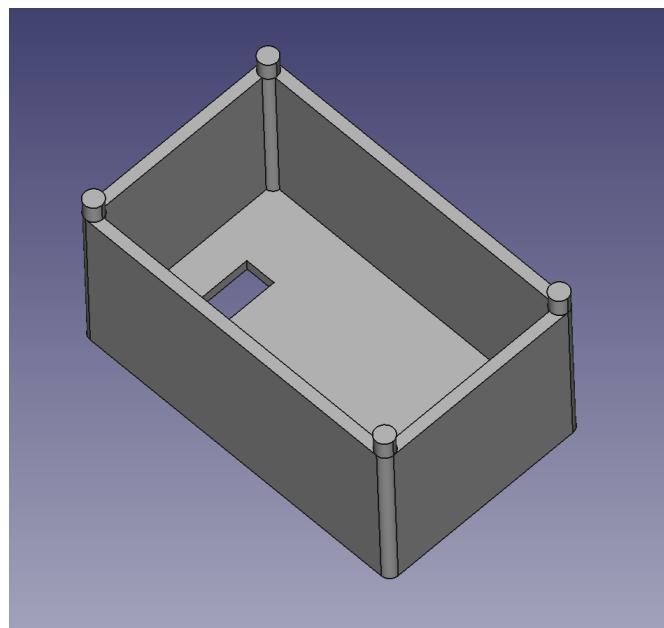
За устройството бе моделиран и създаден корпус, който позволява на сензора да бъде насочван хоризонтално и вертикално, както и да предостави защита срещу прах и напръскване с вода. Първоначалният 3D модел е създаден чрез FreeCAD, софтуер за параметрично 3D моделиране с отворен код, който позволява проектиране на сложни обекти и структури с прецизност и гъвкавост. Моделът бе експортиран като .stl файл и сплайнснат чрез програмата Ultimaker Cura, мощен и удобен за потребителя софтуер за слайсване, който преобразува цифрови 3D модели в инструкции за 3D принтери - gcode.



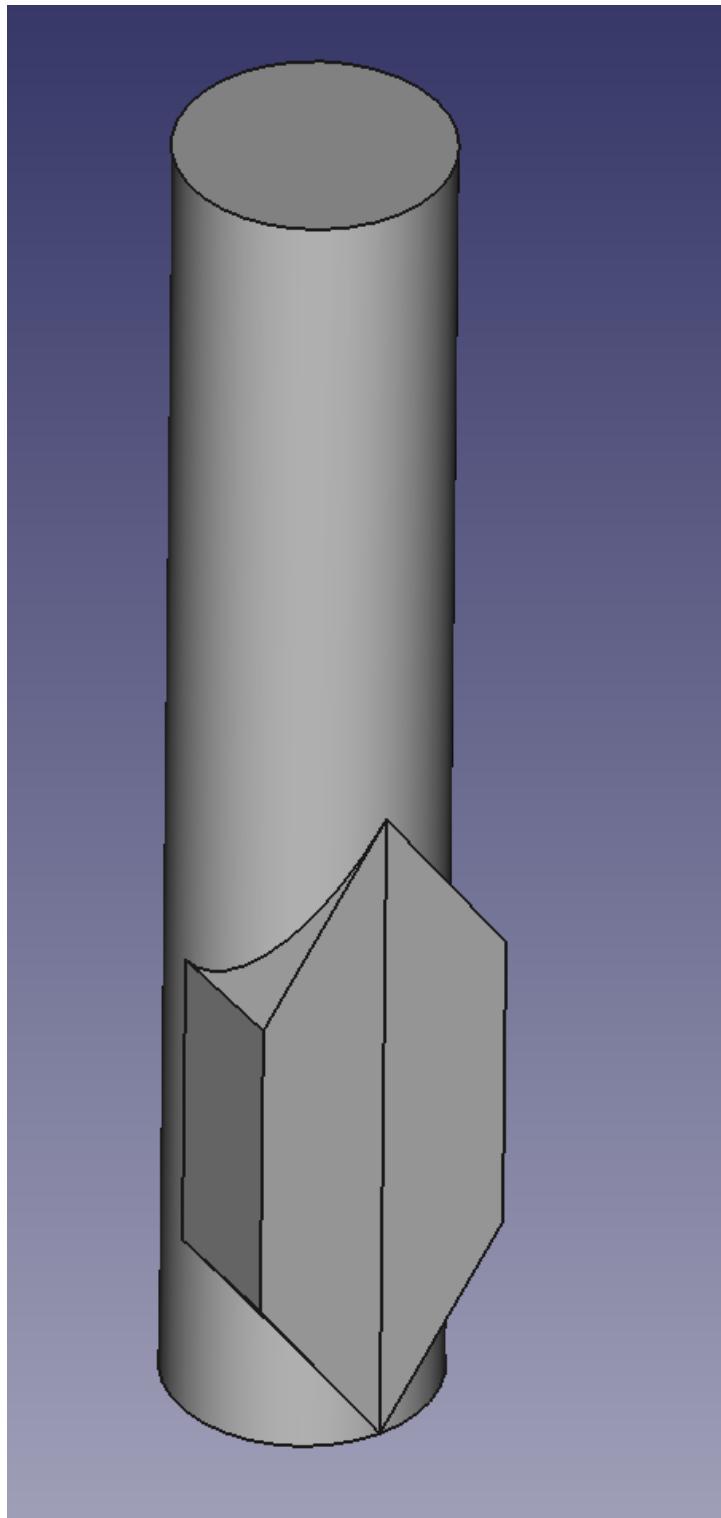
Фигура 6.1: Главна част на корпуса



Фигура 6.2: Главна част на корпуса

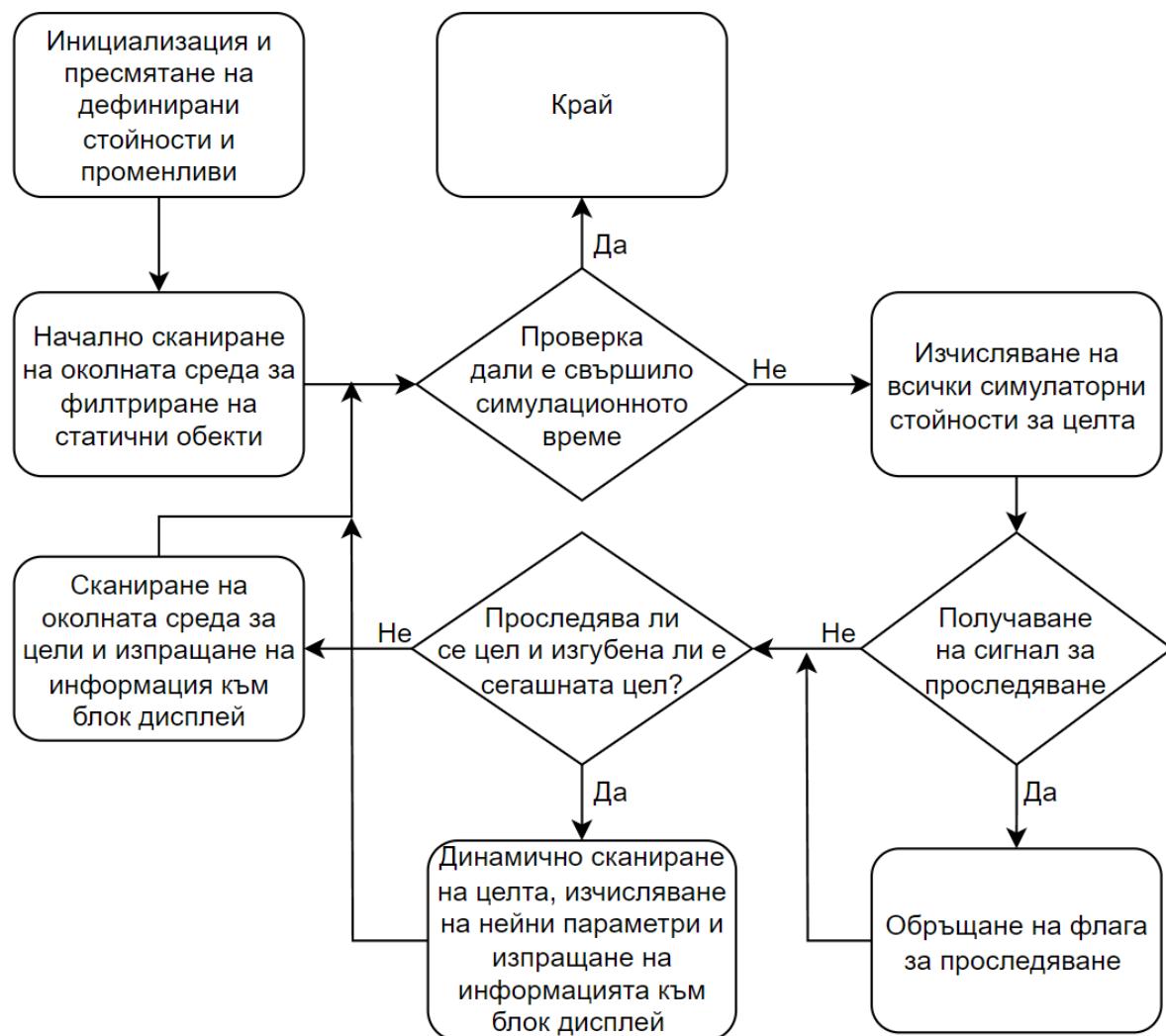


Фигура 6.3: Защитно покритие на електрониката против атмосферни влияния



Фигура 6.4: Ос за монтиране на лидарен сензор

6.2 Блокова схема на симулационен софтуер за системи за търсене и проследяване на цели



Фигура 6.5: Блокова схема на кода

6.3 Симулационен софтуер за системата за търсене и проследяване на цели

```
1 #include <math.h>
2 #include <ESP32Servo.h>
3
4 // Define servo pins (Beta is the vertically turning servo motor)
5 #define SERVO_PIN_ALPHA 3
6 #define SERVO_PIN_BETA 5
7
8 // Define lidar measurements per second and time to be simulated
9 #define tick_rate 15
10 #define simulation_time_ms 25000
11 float cycle_duration_ms = 1000/tick_rate;
12
13 // Define max lidar detection range for very large target in meters, divergence in
   degrees and area to be scanned in degrees
14 // Set vertical scan to the desired furthest up, Set horizontal scan to the
   desired horizontal degrees
15 #define max_lidar_range 250
16 #define lidar_divergence 5
17 #define vertical_degrees_scanned 20
18 #define horizontal_degrees_scanned 60
19
20 // Define the times acquiring needs to not find a target before acquisition is
   canceled and the size of the starting acquisition box is in degrees (The box
   changes size dynamically based on target size)
21 #define acq_width 10
22 #define acq_height 6
23 #define ttl_scans 5
24
25 // Variables used for keeping track of the TTL of the acquisition and the dynamic
   acquisition box
26 int high_acq_edge, low_acq_edge, left_acq_edge, right_acq_edge;
27 int ttl;
28
29 // Arrays to store X, Y, Z coordinates of the target at two different points in
   time, used for data calculation
30 float first_target_coordinates[3];
31 float second_target_coordinates[3];
32
33 // Variables for target data
34 float target_speed, old_target_speed, acceleration, heading;
35
36 // First bool is used for alternating between which X, Y, Z coordinate array is
   being written to, second represents whether the system is tracking a target
37 bool first_scan = true;
38 bool acquiring = false;
39
40 // Calculate largest area that can be scanned without leaving areas unscanned
41 float bar_size = (lidar_divergence / 2) * sqrt(2);
42 float vertical_bar_size = bar_size;
```

```

43
44 // Define values used for servo guidance in degrees (-90 to +90 for horizontal, 0
   to +90 for vertical)
45 float alpha_servo_degrees = 0, beta_servo_degrees = 0;
46 int alpha_servo_bars = 0, beta_servo_bars = 0;
47
48 // Define starting target coordinates (1 unit = 1m)
49 float target_x = 100;
50 float target_y = 0;
51 float target_z = 20;
52
53 // Define speed in m/s for each vector
54 float speed_x = 0.0;
55 float speed_y = 0.0;
56 float speed_z = 0.0;
57
58 // Define target alpha and beta angles (horizontal and vertical angles in
   degrees), and target distance from 0,0,0
59 float target_alpha;
60 float target_beta;
61 float xyMagnitude;
62 float distance;
63
64 // Initialize servo motors
65 Servo servoAlpha;
66 Servo servoBeta;
67
68 // 2D array for storing distances to ground clutter for filtration
69 // IMPORTANT: SIZE MUST BE PRECALCULATED BASED ON DIVERGENCE SO THAT IT ISN'T TOO
   SMALL
70 float ground_clutter[36][18];
71
72 // Sensor's maximum error in meters. Used to prevent returning clutter as a target
   due to imprecision in separate measurements.
73 float deviation = 0.5;
74
75 // Total cycles made
76 int current_cycle = 0;
77 int nextCycleMillis;
78
79 // Bool used to keep track of whether vertical servo motor's rotation direction
80 bool seeking_up = false;
81
82 // Function to calculate the next cycle's millis
83 int calculateNextCycleMillis() {
84   nextCycleMillis = cycle_duration_ms * current_cycle;
85   return nextCycleMillis;
86 }
87
88 // Functions for printing simulator information
89 void printServoDebugInfo() {
90   Serial.print(alpha_servo_degrees);
91   Serial.print(" / ");
92   Serial.println(beta_servo_degrees);

```

```

93 }
94 int printTargetDebugInfo() {
95     Serial.print("[");
96     Serial.print(target_x);
97     Serial.print(", ");
98     Serial.print(target_y);
99     Serial.print(", ");
100    Serial.print(target_z);
101    Serial.print("], ");
102    Serial.print(target_alpha);
103    Serial.print(", ");
104    Serial.print(target_beta);
105    Serial.print(", ");
106    Serial.println(distance);
107 }
108
109 // Void to move servos across search area without leaving spots unscanned.
110 void moveServos(){
111     servoAlpha.write(alpha_servo_degrees);
112     servoBeta.write(beta_servo_degrees);
113
114     alpha_servo_bars = round(alpha_servo_degrees / abs(bar_size));
115     beta_servo_bars = round(beta_servo_degrees / abs(bar_size));
116
117     alpha_servo_degrees += bar_size;
118
119     if (alpha_servo_degrees >= horizontal_degrees_scanned) {
120         bar_size = -bar_size;
121         beta_servo_degrees += vertical_bar_size;
122     } else if (alpha_servo_degrees <= 0) {
123         bar_size = abs(bar_size);
124         beta_servo_degrees += vertical_bar_size;
125     }
126     if (beta_servo_degrees >= vertical_degrees_scanned) {
127         vertical_bar_size = -abs(vertical_bar_size);
128         if (seeking_up){
129             Serial.println();
130             seeking_up = !seeking_up;
131         }
132     } else if (beta_servo_degrees <= 0) {
133         if (!seeking_up){
134             Serial.println();
135             seeking_up = !seeking_up;
136         }
137         vertical_bar_size = abs(vertical_bar_size);
138     }
139 }
140
141 bool targetIllumination(float alpha_servo, float beta_servo, float
142     target_alpha_from_lidar, float target_beta_from_lidar) {
143     // Calculate the absolute differences between the lidar illumination angle and
144     // simulated target angle to lidar
145     float diff1 = abs(alpha_servo - target_alpha_from_lidar);
146     float diff2 = abs(beta_servo - target_beta_from_lidar);

```

```

145
146 // Return whether target is close enough to lidar beam to receive a return
147 return (diff1 <= bar_size && diff2 <= bar_size);
148 }
149
150
151 float lidarMeasurement(){
152 if (targetIllumination(alpha_servo_degrees, beta_servo_degrees, target_alpha,
153 target_beta) && distance < max_lidar_range && distance <
154 (ground_clutter[alpha_servo_bars][beta_servo_bars] - deviation)){
155 return distance;
156 }
157
158 // Function to calculate target coordinates, alpha, beta, and distance
159 void simulateTarget() {
160 // Calculate target_x, target_y, target_z
161 target_x += (speed_x/tick_rate);
162 target_y += (speed_y/tick_rate);
163 target_z += (speed_z/tick_rate);
164
165 // Calculate target_alpha, target_beta and distance
166 target_alpha = atan2(target_x, target_y) * 180.0 / PI;
167 xyMagnitude = sqrt(sq(target_x) + sq(target_y));
168 target_beta = atan2(target_z, xyMagnitude) * 180.0 / M_PI;
169 distance = sqrt(sq(target_x) + sq(target_y) + sq(target_z));
170 }
171
172 float calculateHeight(float distance) {
173 // Convert angles from degrees to radians
174 float verticalAngleRad = radians(beta_servo_degrees);
175 float horizontalAngleRad = radians(alpha_servo_degrees);
176
177 // Calculate the height (Z-coordinate) of the point
178 float height = distance * tan(verticalAngleRad);
179
180 return height;
181 }
182
183 void setup() {
184 Serial.begin(115200);
185 servoAlpha.attach(SERVO_PIN_ALPHA);
186 servoBeta.attach(SERVO_PIN_BETA);
187 // Initial scan for ground clutter filtration
188 while (beta_servo_degrees <= vertical_degrees_scanned || alpha_servo_degrees <=
189 horizontal_degrees_scanned){
190 moveServos();
191 ground_clutter[alpha_servo_bars][beta_servo_bars] = lidarMeasurement();
192 delay(cycle_duration_ms);
193 }
194
195 void targetAcquisition(){
```

```

196 int high_target_edge = 0, low_target_edge = 0, left_target_edge = left_acq_edge,
       right_target_edge = 0;
197 for (int y_degrees = low_acq_edge; y_degrees <= high_acq_edge; y_degrees += bar_size){
198     for (int x_degrees = left_acq_edge; x_degrees <= right_acq_edge; x_degrees += bar_size){
199         servoAlpha.write(x_degrees);
200         servoBeta.write(y_degrees);
201         float lidar_distance_to_target = lidarMeasurement();
202         if (lidar_distance_to_target < max_lidar_range){
203             ttl = ttl_scans;
204             low_target_edge = y_degrees;
205             if(right_target_edge < x_degrees){
206                 right_target_edge = x_degrees;
207             }
208             if(high_target_edge < y_degrees){
209                 high_target_edge = y_degrees;
210             }
211             if(left_target_edge > x_degrees){
212                 high_target_edge = x_degrees;
213             }
214             if (first_scan){
215                 first_scan = !first_scan;
216                 first_target_coordinates[0] = lidar_distance_to_target * cos(x_degrees) *
217                     sin(y_degrees);
218                 first_target_coordinates[1] = calculateHeight(lidar_distance_to_target);
219                 first_target_coordinates[2] = lidar_distance_to_target * sin(x_degrees) *
220                     sin(y_degrees);
221             }
222         else{
223             first_scan = !first_scan;
224             second_target_coordinates[0] = lidar_distance_to_target * cos(x_degrees) *
225                 sin(y_degrees);
226             second_target_coordinates[1] = calculateHeight(lidar_distance_to_target);
227             second_target_coordinates[2] = lidar_distance_to_target * sin(x_degrees) *
228                 sin(y_degrees);
229         }
230     }
231     old_target_speed = target_speed;
232     target_speed = sqrt(sq(second_target_coordinates[0] -
233         first_target_coordinates[0]) + sq(second_target_coordinates[1] -
234         first_target_coordinates[1]) + sq(second_target_coordinates[2] -
235         first_target_coordinates[2])) / cycle_duration_ms;
236     acceleration = target_speed - old_target_speed;
237
238     heading = atan((second_target_coordinates[2] - first_target_coordinates[2]) /
239         (second_target_coordinates[0] - first_target_coordinates[0]));
240     if ((second_target_coordinates[2] - first_target_coordinates[2]) < 0 &&
241         (second_target_coordinates[0] - first_target_coordinates[0]) >= 0) {
242         heading += PI;
243     } else if ((second_target_coordinates[2] - first_target_coordinates[2]) < 0
244         && (second_target_coordinates[0] - first_target_coordinates[0]) < 0) {

```

```

237     heading -= PI;
238 }
239
240 Serial.print("Speed: ");
241 Serial.print(target_speed);
242 Serial.print(" Height: ");
243 Serial.print(calculateHeight(lidar_distance_to_target));
244 Serial.print(" Acceleration: ");
245 Serial.print(acceleration);
246 Serial.print(" Heading: ");
247 Serial.println(degrees(heading));
248 }
249
250 high_acq_edge = high_target_edge + bar_size;
251 low_acq_edge = low_target_edge - bar_size;
252 left_acq_edge = left_target_edge - bar_size;
253 right_acq_edge = right_target_edge + bar_size;
254 }
255 }
256
257 void loop() {
258 // Wait until it's time to start the cycle
259 while (millis() <= calculateNextCycleMillis()) {}
260
261 // End simulation when the simulation time has passed
262 if (millis() >= simulation_time_ms){
263   Serial.println("Simulation has finished.");
264   while (true){}
265   delay(1000);
266 }
267
268 simulateTarget();
269
270 if (Serial.available() >= 5) {
271   if (!acquiring){
272     ttl = ttl_scans;
273     acquiring = !acquiring;
274     low_acq_edge = Serial.parseInt() - (acq_height/2);
275     Serial.read();
276     left_acq_edge = Serial.parseInt() - (acq_width/2);
277     Serial.read();
278     right_acq_edge = left_acq_edge + acq_width;
279     high_acq_edge = low_acq_edge + acq_height;
280   } else {acquiring = !acquiring;}
281 }
282
283 if (ttl > 0 && acquiring){
284   ttl--;
285   targetAcquisition();
286 }else {
287   moveServos();
288   float lidar_distance_to_target = lidarMeasurement();
289   if(ground_clutter[alpha_servo_bars][beta_servo_bars] <=
        lidar_distance_to_target + deviation){

```

```
290     lidar_distance_to_target = max_lidar_range;
291 }
292 if (lidar_distance_to_target < max_lidar_range){
293     Serial.print(lidar_distance_to_target);
294     Serial.print(",");
295 }else {Serial.print("0");}
296 }
297 //printServoDebugInfo();
298 //printTargetDebugInfo();
299 //Serial.println(millis());
300 current_cycle += 1;
301 }
```

Програмен код 6.3: Пълен софтуер за симулация на работата на радар.

ЗАКЛЮЧЕНИЕ

В тази дипломна работа са разгледани и анализирани съществуващи решения за търсене и проследяване на въздушни цели и се установи че предлаганите военни и професионални решения са твърде скъпи, бавни за разполагане и не предоставят достатъчна точност на измерванията за да позволят изчисляване на огнево решение срещу малки цели, като дронове. Също така е установено че любителските решения са лесни и евтини за създаване, но не предлагат почти никаква гъвкавост в начина на използване и събраните данни от тези системи могат единствено да се използват за предупреждение. Разгледани са, сравнени са и са избрани устройства и технологии за създаване на подобна система. Чрез тях е създадена система, която решава проблемите на съществуващите системи. Точността на измерванията е няколко сантиметра, докато при съществуващите системи е няколко метра. Създаденият упървляващи софтуер използва тези измервания и изчислява всички стойности относно текущата и бъдещата позиция на проследяваната цел в пространството и ги представя на оператора в лесно разбираем вид. Също така бе създаден и симулационен софтуер, чрез който може да се симулира работата на подобна система. Този софтуер позволява използването на спецификациите на симулирания сензор, както и да се задават от потребител множество настройки на цел в симулираната среда.

По време на проучването, много трудно бе намерен рентабилен сензор за измерване на разстояние. Със по-голям бюджет, разработеното устройство би било много по-потентно и би имало много повече реални приложения.

В бъдеще, тази дипломна работа би могла да бъде развита като се добави начин за комуникация между модули и споделяне на информация за цели между тях и други активи. Също така може да бъде интегриран прихващащ като например балистична зенитна ракета за по-далечни разстояния или оръдие, за по-къси. За тях може и да бъде разработена система за автоматично презареждане и различни бойни глави.

ИЗПОЛЗВАНИ СЪКРАЩЕНИЯ

IFF - Identification Friend or Foe

LWIR - Long-Wave Infrared

I2C - Inter-Integrated Circuit

SPI - Serial Peripheral Interface

UART - Universal Asynchronous Receiver-Transmitter

IDE - Integrated Development Environment

GPIO - General Purpose Input/Output

USB - Universal Serial Bus

SRAM - Static Random Access Memory

LCD - Liquid Crystal Display

EEPROM - Electrically Erasable Programmable Read-Only Memory

FMCW - Frequency Modulated Continuous Wave

SMD - Surface Mount Device

TFT - Thin-Film Transistor

LED - Light Emitting Diode

IoT - Internet of Things

ИЗПОЛЗВАНА ЛИТЕРАТУРА

- [1] https://www.youtube.com/watch?v=ZJE5fwrZzXY&ab_channel=Electrobrains
- [2] https://www.youtube.com/watch?v=ZJE5fwrZzXY&ab_channel=Electrobrains
- [3] <https://www.dedrone.com/optical-sensors-for-drone-detection>
- [4] <https://missiledefenseadvocacy.org/defense-systems/an-mpq-64-sentinel/>
- [5] <https://randomnerdtutorials.com/getting-started-with-esp32/>
- [6] <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>
- [7] <https://docs.arduino.cc/hardware/uno-rev3/>
- [8] <https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html>
- [9] <https://support.garmin.com/en-US/?partNumber=010-01722-00&tab=manuals>
- [10] <https://support.garmin.com/en-US/?partNumber=010-02022-00&tab=manuals>
- [11] <https://rfbeam.ch/product/v-ld1-distance-sensor/>
- [12] <https://rfbeam.ch/product/v-ld1-evaluation-kit-rfbeam-microwave/>
- [13] https://www.alldatasheet.com/view.jsp?Searchword=Mg996r&gad_source=1&gclid=CjwKCAiAloavBhBwE
- [14] http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf
- [15] <https://www.electromate.com/mechatronic-automation/mechatronic-automation-components/stepper-motors/nema-stepper-motors/nema-17-stepper-motors/>
- [16] <https://www.adafruit.com/product/2050>
- [17] <https://www.dfrobot.com/product-51.html>
- [18] <https://www.britannica.com/technology/laptop-computer>
- [19] <https://www.cei.washington.edu/research/energy-storage/lithium-ion-battery/>
- [20] <https://micro.magnet.fsu.edu/electromag/electricity/batteries/alkaline.html>
- [21] <https://byjus.com/physics/pulse-width-modulation/>
- [22] <https://www.spiceworks.com/tech/tech-general/articles/universal-serial-bus/>
- [23] <https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication.html>
- [24] <https://www.freecad.org/>
- [25] <https://www.kicad.org/>
- [26] <https://code.visualstudio.com/>

- [27] <https://csapp.cs.cmu.edu/3e/docs/chistory.html>
- [28] <https://docs.python.org/3/faq/general.html#what-is-python>
- [29] <https://en.wikipedia.org/wiki/C%2B%2B>

Съдържание

УВОД	1
ПЪРВА ГЛАВА	2
1.1 Проучване на съществуващи системи за сканиране и проследяване на цели	2
1.1.1 Комерсиални системи	2
1.1.2 Военни системи	6
1.1.3 Непрофесионални системи	7
1.1.4 Сравнение и анализ на съществуващи системи	8
1.2 Устройства избрани за дипломния проект	9
1.2.1 Микроконтролер	9
1.2.2 Сензор за измерване на разстояние	16
1.2.3 Мотори за насочване на сензора за измерване на разстояние	19
1.2.4 Дисплей и управление	21
1.2.5 Източник на захранване	24
1.3 Използвани технологии, стандартни и протоколи за комуникация	26
1.3.1 Широчинно импулсна модулация (PWM)	26
1.3.2 Универсална серийна шина (USB)	26
1.3.3 I2C	26
1.3.4 UART	27
1.4 САПР използвани в дипломния проект	28
1.4.1 FreeCAD	28
1.4.2 KiCAD	29
1.5 Програмна среда и програмни езици използвани в дипломния проект	30
1.5.1 Visual Studio Code	30
1.5.2 Програмен език C	31
1.5.3 Програмен език Python	32
1.5.4 Програмен език C++	33
ВТОРА ГЛАВА	34
2.1 Основни изисквания към дипломния проект	34
2.2 Блокова схема на устройството	34
2.2.1 Описание на блоковата схема на устройството	35
ТРЕТА ГЛАВА	36
3.1 Проектиране на принципна електрическа схема на системата за търсене и проследяване на цели	36
3.1.1 Избор на компоненти	36
3.1.2 Принципна електрическа схема	37
ЧЕТВЪРТА ГЛАВА	38
4.1 Информация за печатната платка	38

4.2 Проектиране на печатна платка за системата за търсене и проследяване на цели	40
4.2.1 Графични изображения на корпуси на елементи и триизмерни модели на използваните компоненти	40
4.2.2 Слоеве на печатната платка	41
ПЕТА ГЛАВА	44
5.1 Блокова схема на управляващ софтуер за системата за търсене и проследяване на цели	44
5.2 Управляващ софтуер за системата за търсене и проследяване на цели	45
5.3 Блокова схема на управляващ софтуер за визуализиране на информация и управляване на системата	52
5.4 Управляващ софтуер за визуализиране на информация и управляване на системата	53
ШЕСТА ГЛАВА	55
6.1 Корпус на устройството	55
6.2 Блокова схема на симулационен софтуер за системи за търсене и проследяване на цели	58
6.3 Симулационен софтуер за системата за търсене и проследяване на цели	59
ЗАКЛЮЧЕНИЕ	66
ИЗПОЛЗВАНИ СЪКРАЩЕНИЯ	67
ИЗПОЛЗВАНИ ЛИТЕРАТУРА	68