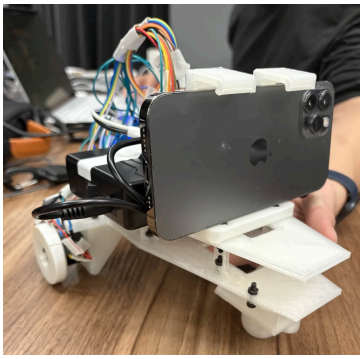
Olivia Ye's Portfolio

olivia.ye@uwaterloo.ca
(415) 481-2309
www.linkedin.com/in/olivia-w-ye
www.github.com/oliviawxye

Table of Contents

iPhone-based Open Source Autonomous Mobile Robot

Github Repository | For Mechatronics Engineering Class of 2026 Fourth Year Capstone Project *2025 - Present*



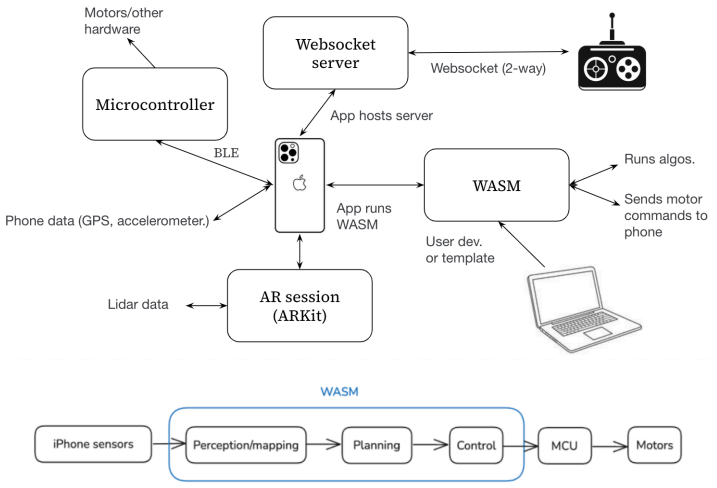
Prototype of the robot

What?

- Building an **open-source, autonomous mobile robot development platform** capable of running **SLAM** that utilizes an **iPhone** for sensors and compute, in a team of 5
- Focused on high reproducibility, low-cost, and software and hardware extensibility

Why?

- Roboticists face a **cost-capability tradeoff when selecting robotics platforms**—capable options like the Turtlebot 4 are expensive (~$1000 CAD for sensors and compute alone), while cheaper alternatives like LEGO SPIKE offer limited capability
- Many users have already invested in smartphones, which include high-quality cameras, positional sensors, and LiDAR—making them ideal for repurposing as robotics platforms

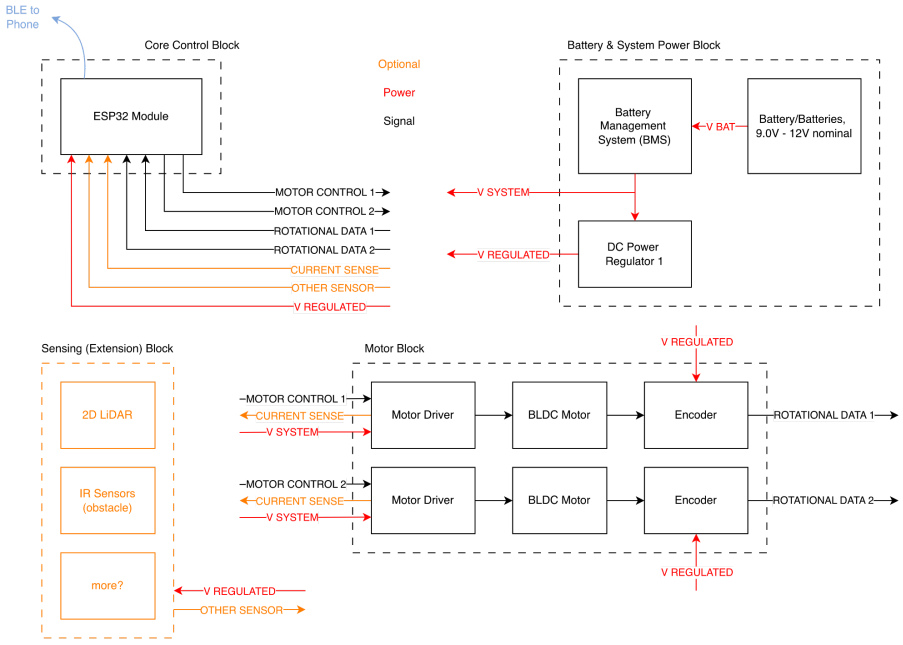


Software block diagram & autonomy block diagram within the WASM module

How?

- **Apple's ARKit** provides LiDAR data access
- **WebAssembly (WASM)** serves as the user-facing development method due to its language-agnostic binary instruction format, enabling roboticists to develop in their preferred programming language
- An **iOS app** exposes an **API** that allows the user's WASM module to access data from cameras, LiDAR, IMU, and GPS
- The **autonomy stack resides within the WASM module**, which roboticists can upload and run in the app, abstracting away iOS development complexity
- The module outputs motor commands to the app, which are transmitted to the hardware via **Bluetooth Low Energy (BLE)**

- Core control uses an **ESP32 microcontroller**, which supports BLE communication and provides sufficient GPIO
- **Two-wheel differential drive** configuration selected for simple kinematics and dynamics
- Initial design uses **brushless DC (BLDC) motors with encoders and field-oriented control (FOC)**; however, motors can be swapped as all components are modular
- Initial design relies solely on iPhone sensors, but offers **hardware extensibility** for external sensors
- On-board **12V battery** powers the microcontroller and motors

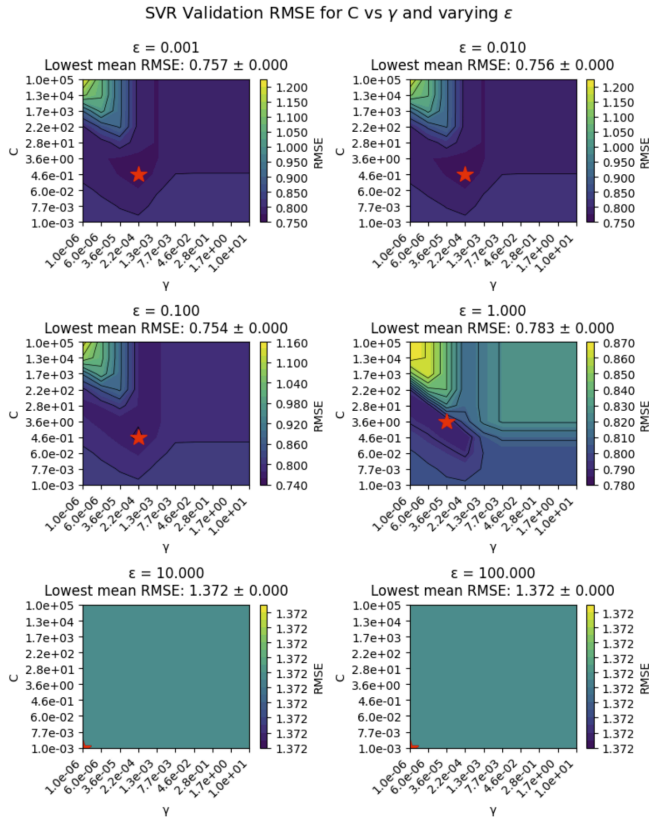


Hardware block diagram

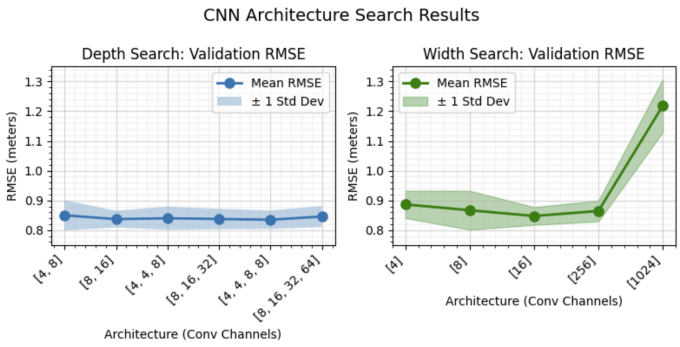
Estimating Depth from Monocular Vision using AI

Github Repository | Paper | For course SYDE 522: Foundations of AI *2025*

Example of a RGB photo and depth map from the NYU Depth V2 dataset



Tuning hyperparameters for support vector regression (SVR)



Tuning convolutional channels for convolutional neural networks (CNN)

What?

- Investigated and compared **four machine learning architectures** (polynomial regression, support vector regression (SVR), multi-layer perceptrons (MLP), and convolutional neural networks (CNN)) to **estimate mean scene depth from monocular RGB images** using the **NYU Depth V2 dataset** (1449 samples)
- Achieved sub-meter accuracy (0.78m RMSE) with SVR

How?

- **Downsampled** RGB images and depth maps from 640×480 to 40×30 pixels to maintain spatial complexity while reducing the feature space, addressing the **limited sample-to-feature ratio**
- Systematically **tuned hyperparameters** for each model using 10-iteration validation with **80-20** train-validation splits
  - Regression: *n* and regularization
  - SVR: regularization, insensitivity, and RBF kernel
  - MLP: # epochs and depth/width of hidden layers
  - CNN: # epochs and depth/width of convolutional layers
- Regression: switched from Ridge regression to SGDRegressor within **scikit-learn** with standard scaling and batch processing to handle the high-dimensional feature space within computational constraints
- MLP & CNN: Designed NN experiments with **PyTorch** to explore depth versus width tradeoffs across architectures ranging from narrow-deep networks (4-8 neurons/channels) to wide-shallow networks (256-1024 neurons/channels), using **Adam optimizer** with **dropout regularization**
- Benchmarked all models using **root-mean-squared error (RMSE) on identical test data**, comparing classical machine learning approaches against deep learning methods

Key Learnings

- The investigation revealed significant **over-parameterization**, with the simplest MLP containing 4859 parameters trained on only 927 samples (~5:1 ratio); **data augmentation techniques**—such as horizontal flipping, random cropping with adjusted depth maps—could augment the dataset to achieve the recommended 10 samples/parameter while preserving depth relationships

Model	Parameters	RMSE [m]
Regression	$n = 2, \lambda = 4830$	2.1155
SVR	$C = 0.46, \gamma = 2.15 \times 10^{-4}, \epsilon = 0.1$	0.7816
MLP	[256]	0.8277
CNN	[4, 4, 8, 8]	0.7974

Testing RMSE results

Sensor Synchronization of LiDAR and Cameras to GPS

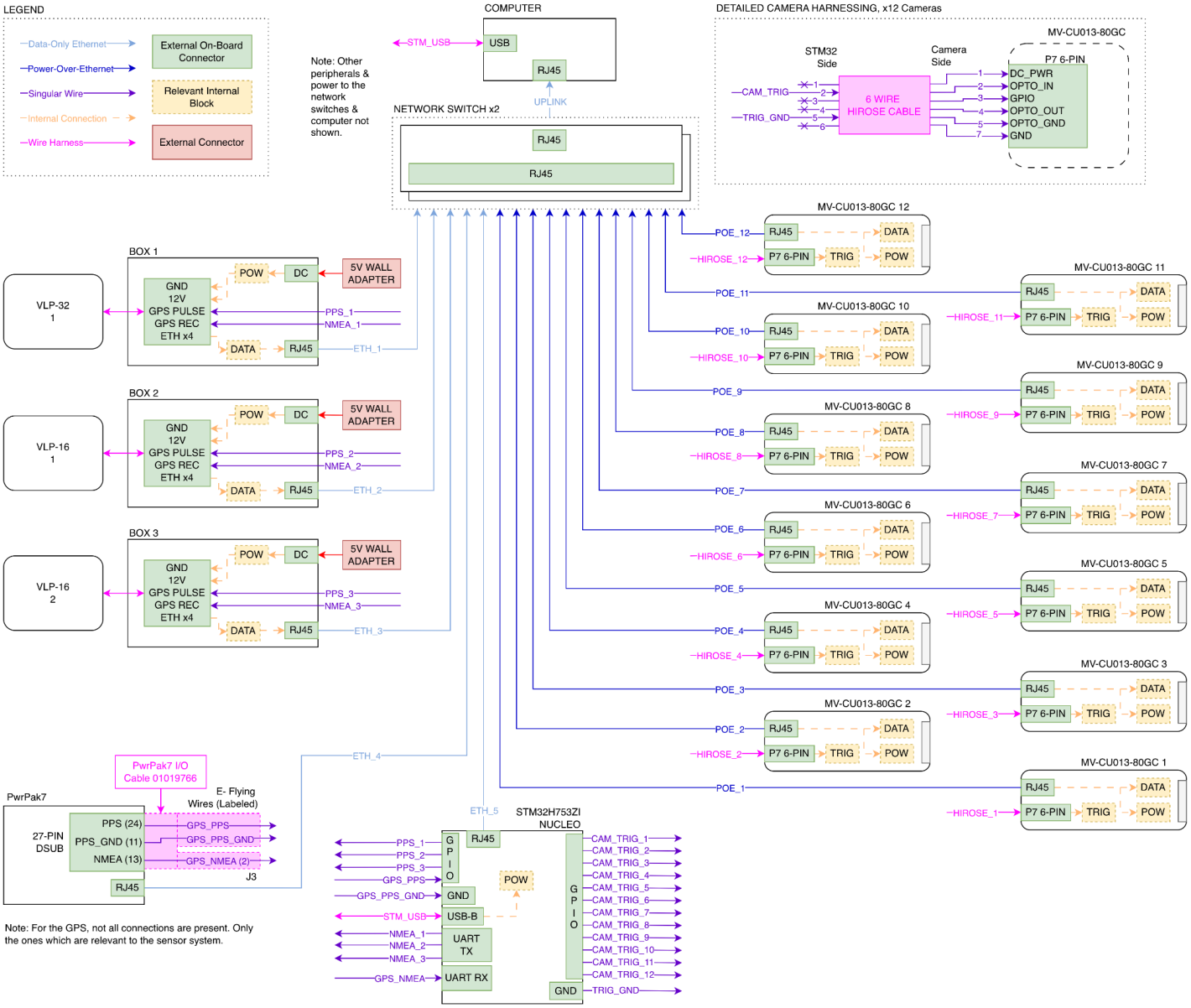
Github Repository | For Watonomous *2025 - Present*

What?

- Implementing **sub-millisecond timing and synchronization accuracy** between the vehicle's computer, GPS, and external sensors including **1 VLP-32 LiDAR, 2 VLP-16 LiDARs, and 12 global shutter CMOS cameras**

How?

- Used **NovAtel's PwrPak7 GPS** as the **Precision Time Protocol (PTP)** grandmaster, propagating synchronized time through the **Ethernet** network to the GPS, cameras, and computer
- The **STM32 Nucleo** receives the GPS Pulse Per Second (PPS) signal and **forwards it as hardware triggers to the cameras**, which capture frames upon receiving the trigger and timestamp the data using the **PTP-synchronized clock**
- The **STM32 Nucleo** receives and **forwards both GPS PPS and NMEA signals to the LiDARs** to enable accurate timestamping of scans—NMEA provides contextualized time information (date, time, timezone) and PPS provides sub-millisecond timing accuracy



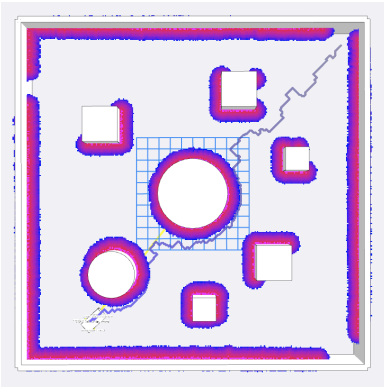
Communication and harnessing diagrams linking the computer, LiDARs, cameras, and GPS

Navigation of a 2-W Mobile Robot in Simulation using C++

Github Repository | For Watonomous: Autonomy Team Assignment *2025*

What?

- Implemented complete autonomous navigation stack for a 2-wheel differential drive mobile robot in **simulation**, enabling **point-to-point navigation** while avoiding static obstacles using **laser scanner perception**
- Developed **four interconnected ROS2 nodes in C++:** Costmap generation from laser scans, global map memory with pose fusion, A* path planning, and Pure Pursuit trajectory control
- Built and deployed the system using **ROS2 Humble**, **Gazebo** simulation, and **Foxglove** for visualization, containerized in **Docker** for reproducibility



Visualization of global occupancy map and planned path

How?

- Costmap Node: Processed laser scan data to create **occupancy grids** by converting distance measurements to grid coordinates, marking obstacles, and inflating boundaries to maintain safe clearance around objects
- Map Memory Node: Built a global map by **combining local costmaps with odometry data**, updating the map only when the robot moved 1.5m to reduce computational load
- Planner Node: Used **A* algorithm** to find optimal paths through the occupancy grid from start to goal, calculating costs based on distance traveled and estimated distance remaining, using the **Euclidean heuristic**
- Control Node: Implemented a **Pure Pursuit controller** to follow planned paths by selecting lookahead points ahead of the robot and calculating the steering needed to reach them
- Connected all nodes using **ROS2 publishers, subscribers, and timers** to enable **real-time communication** between perception, mapping, planning, and control modules

Localization, Planning, and Control of Turtlebot 4 using Python

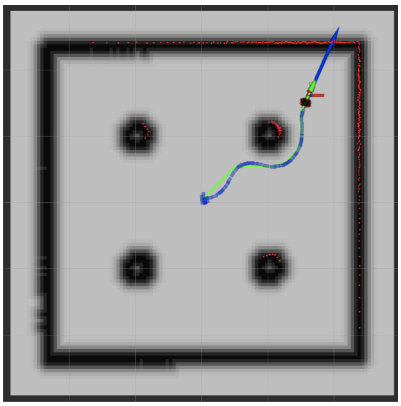
Github Repository | For course MTE 544: Autonomous Mobile Robots *2025*

What?

- Implemented autonomous navigation system for a Turtlebot 4 using particle filter localization, A* graph search, and trajectory tracking control in **Python** with **ROS2 Humble** and **RViz**, in a team of 3
- Developed A* path planner with configurable heuristics and PID linear and angular movement control
- Used SLAM-based map acquisition and validated navigation performance on physical hardware

How?

- SLAM Map Acquisition: Used laser scanner data with Turtlebot's **SLAM** algorithms to build 2D occupancy grid maps
- Planner: Used **A* algorithm** to find optimal paths through the occupancy grid from start to goal, calculating costs based on distance traveled and estimated distance remaining, testing both **Euclidean and Manhattan heuristics**
- Cost Map Processing: **Transformed occupancy grids into cost maps**, inflated obstacle boundaries for clearance, and discretized continuous space for graph-based planning
- **Particle Filter Localization**: Estimated robot pose by propagating particles with motion model, weighted particles based on laser scan likelihood compared to map data, and resampled to concentrate particles around high-probability poses
- Trajectory Execution: Converted A* waypoint list into sequential goal poses, used **PID control** to track each waypoint by calculating velocity commands, and published Twist messages to /cmd_vel topic for **differential drive control**



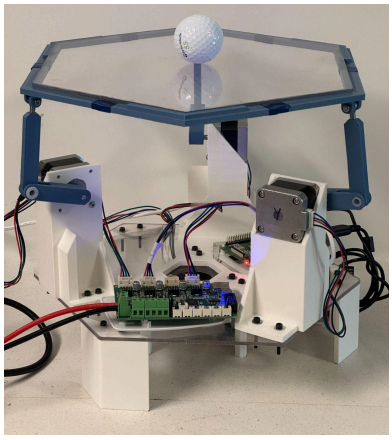
Particle filter localization visualization in RViz with A* path planning and robot tracking

Automatic Self-Balancing 3-DOF Stewart Platform

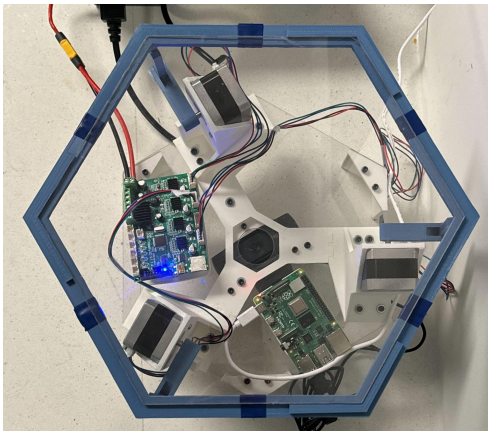
Github Repository | For course MTE 380: Mechatronics Design Workshop *2024*

What?

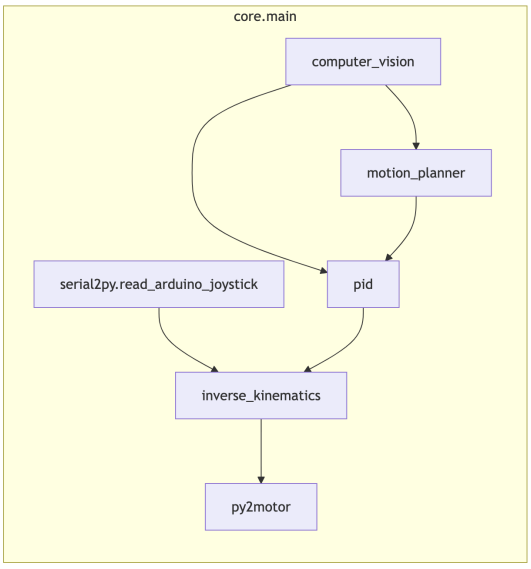
- Created a portable Stewart Platform capable of **moving a ball in three degrees of freedom** (roll, pitch, z-axis) in a team of 4
- The autonomous system operates with a ping pong ball, golf ball, and metal bearing on a **25 cm × 25 cm platform**, contained within a 50 cm^3 form factor
- Capable of reliably **balancing the ball** at a setpoint and rejecting external disturbances
- Capable of reliably moving the ball along **pre-planned trajectories**



Platform balancing a golf ball



Top-down view

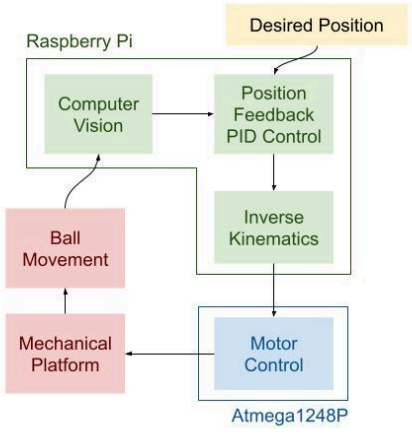


Main control flow

How?

- Developed **OpenCV** code in **Python** using **Canny edge detection** and **Hough transforms** to detect the ball position on the platform from a USB webcam connected to a Raspberry Pi
- Tuned **PID position feedback** control for ball balancing:
  - **Proportional**: Ensures the system quickly responds to disturbances
  - **Integral**: Corrects steady-state error so the ball converges at the desired position
  - **Derivative**: Reduces oscillations and ensures a quick settling time
- Solved **inverse kinematics** using **rotation and translation transforms** on the vector relationships between the three motor axes and mating points on the platform to translate platform angles to motor angles
- Developed **two planners** capable of rejecting external disturbances: (1) consistently returning the ball to the platform center, and (2) moving the ball in a triangular trajectory around the platform

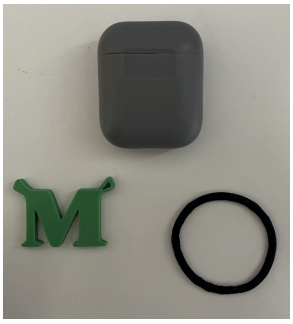
- Repurposed an Ender3 3D printer board with an **Atmega1284P** microcontroller and **A4988 drivers** to control three stepper motors, using **MultiStepper for Arduino in C++**
- Developed **synchronized** motor control - where all three motors start and stop at the same time, regardless of the angle that each motor has to travel
- Used **Nema17 rotational stepper motors** to achieve precise angular rotation without the need for external feedback sensors for motor positions
- Designed all components for FDM 3D printing with friction and clearance tolerances using **SolidWorks**, applying **Design for Manufacturing/Assembly (DFM/DFA)** principles for rapid prototyping and iteration
- Total system cost including hardware and electronics remained below $500



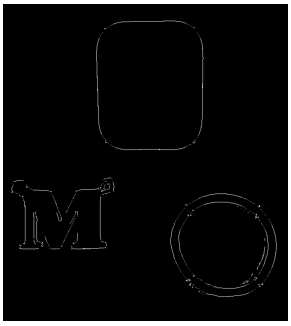
System Overview

Edge Detection and Image Segmentation Program

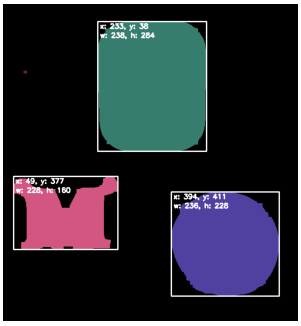
Github Repository | For course MTE 203: Advanced Calculus *2023*



Simple objects on a white background

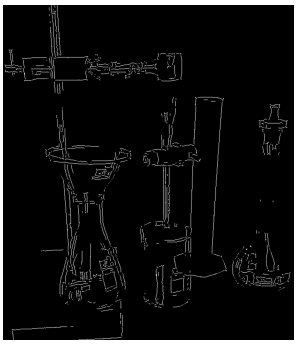


Clear edges after detection

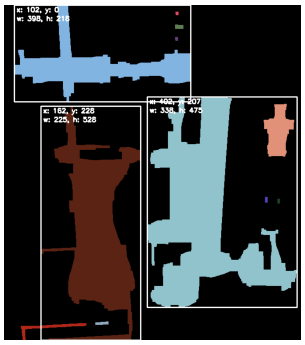


Simply segmented and separated objects

Objects on a colored background



Clear edges after detection



Largest contours connected and filled

What?

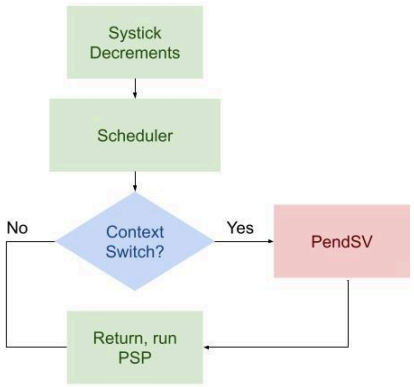
- Created a step-by-step canny edge detection solver in **Python** with **OpenCV**
- Further combined the edge detection by integrating simple **image segmentation** using the largest contours and identifying **Regions of Interest (ROIs)**

How?

- Applied noise filtering using **Gaussian blur**
- Used the **Sobel operator** to find the intensity gradient of the image in both axes
- Used **lower bound cut-off suppression**, **double (high and low) thresholding**, and **edge tracking by hysteresis** using **empirically tuned bounds** to filter noise and emphasize the true edges
- Integrated **morphological operations** to close and fill the large contours to highlight large objects
- Annotated **bounding boxes** around significant objects to identify ROIs

Real-Time Operating System Kernel on an STM32 ARM Cortex M4

Github Repository - Kernel | Github Repository - FreeRTOS | For course MTE 241: Computer Structures and RTOS *2023*



High-level overview of the Systick interrupt flow for scheduling

What?

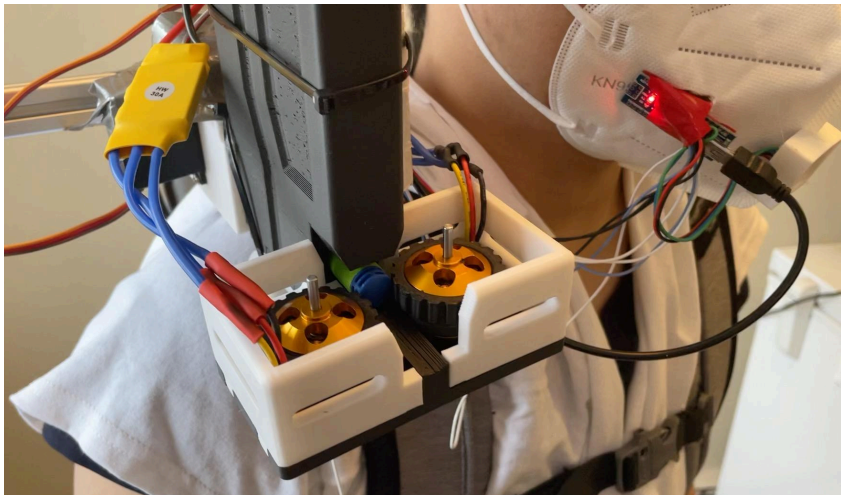
- Created a firm RTOS kernel from scratch in **C** on an **STM32 Cortex M4**, following the **ARM architecture**
- Re-created a second implementation using **FreeRTOS**

How?

- Implemented **pre-emptive schedulers** using **Systick**, runtime, deadlines, **priority**, and software **interrupt handling** with Interrupt Service Routines (ISRs) to enable concurrency and multi-threading emulation on a **single-core CPU**
- Enabled smooth context switching with a **thread control block** to keep track of the thread's registers (including arguments, local variables), stack, and current state
- **Prevented race conditions** in resource management during inter-thread communication using synchronization (**mutexes and semaphores)** for mutual read/write resources
- Developed **kernel helper processes** to track stack usage, thread runtime and deadlines, and number of execution cycles

Anti Anti-Masker Mask

Github Repository | Devpost | For University of Toronto Hardware Hackathon: Make UofT *2022*



Assembled prototype of the shooter and face mask

What?

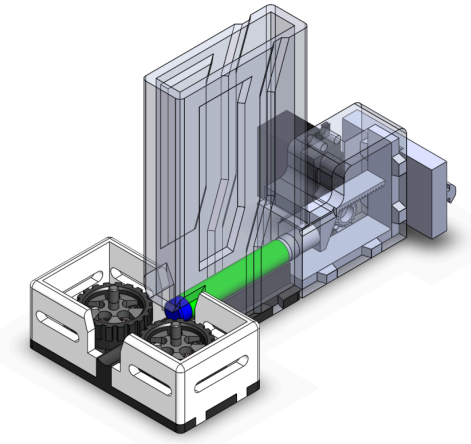
- Created the Anti Anti-Masker Mask in a team of 4—a **wearable** defense mechanism to protect against COVID-19 mandate violators
- The wearable system detects whether a person within a 6-foot forward vicinity is wearing a mask and fires darts at unmasked individuals until they move beyond 6 feet away

Results?

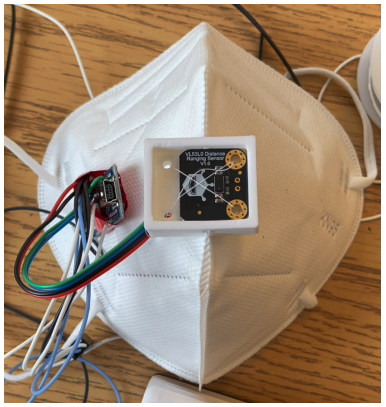
- Placed **second** overall at MakeUOfT, U of Toronto's 24-hour **hardware hackathon**, out of **48 submissions and 180+ participants**

How?

- Integrated a **Time-of-Flight (ToF) sensor** and **live IP camera stream** for perception
- Detected **bounding boxes of faces** within the camera frame using **OpenCV** in **Python**
- Activated shooting functionality when faces were detected while the ToF sensor confirmed objects within the 6-foot range
- Rapidly prototyped a compact and wearable nerf-dart shooter using **SolidWorks** in 8 hours, applying **Design for Manufacturing/Assembly (DFM/DFA)** principles
- Designed a dart feeding system using a **rack-and-pinion** mechanism with a **servo** to dispense darts into a flywheel-based firing mechanism powered by **brushless DC (BLDC) motors**
- Soldered and assembled the **Arduino**-controlled electromechanical system
- Powered the system using **LiPo batteries** with **electronic speed controllers (ESCs)** to drive the BLDC motors

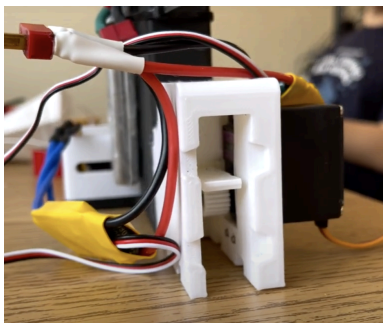


SolidWorks CAD of the shooter



TOF Sensor and Arduino

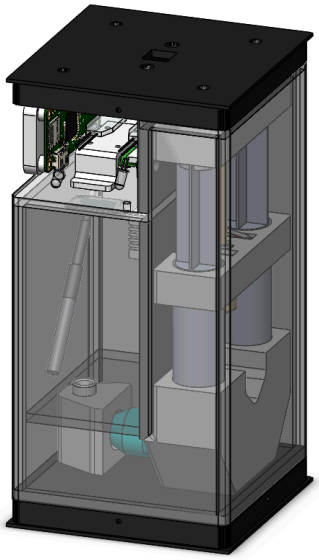
IP Camera Stream



Rack and pinion dart feeder

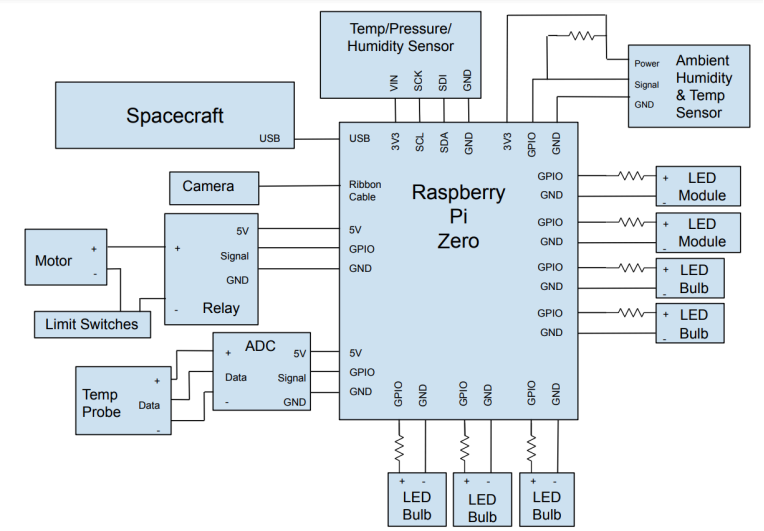
Microgravity Payload for Blue Origin Launch NS-23

Github Repository | News | For Shad Canada and Blue Origin *2020 - 2022*



SolidWorks CAD

Fully assembled payload



Wiring layout of the Raspberry Pi-based embedded system

What?

- Designed, manufactured, and assembled a **four-by-four-by-eight inch payload** to "Investigate Polyurethane Foam in Microgravity" that mixes and aerates a two-component foam during three minutes of zero gravity
- My team of 10 was selected to build the physical payload out of 60 teams and 600+ participants as the **winner of the Shad Canada 2020 Microgravity Design Challenge**

How?

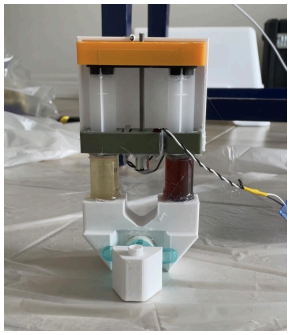
- Designed and soldered a complete electrical system to control **actuators** and receive input from a **camera** and poll **temperature, pressure, and humidity sensors** via **I2C** and **1-Wire** with a **strict power budget of 5V and 0.9A**
- Developed a robust and modulated software system with **Python** on **Raspberry Pi** and **Linux** to parse **serial** flight telemetry and control the payload
- Rapidly prototyped and iterated a compact syringe ejection system with a linear actuator and limit sensors with **SolidWorks, FDM 3D printing, and DFA principles**
- Wrote the Payload Design proposal with **electrical schematics, CAD drawings, and BOM**

Results?

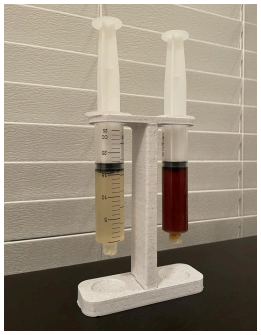
- Built the **first Canadian student payload** launched with Blue Origin's New Shepard Rocket
- Created a microgravity payload that further contributes to the scientific inquiry initially found from the European Space Agency's 2009 study on Foam Stability
- Intended to create solid foam in space on the basis that foam in microgravity has increased stability due to uniform bubble formation in the absence of gravity

- Submitted a paper to the **International Astronautical Federation** for the **2021 International Astronautical Congress** detailing the project's progress including the initial concept, ground testing, design iterations, and outreach

Video stream of the foam mixing chamber



Syringe subsystem



Testing components

NS-23 launch with payload inside