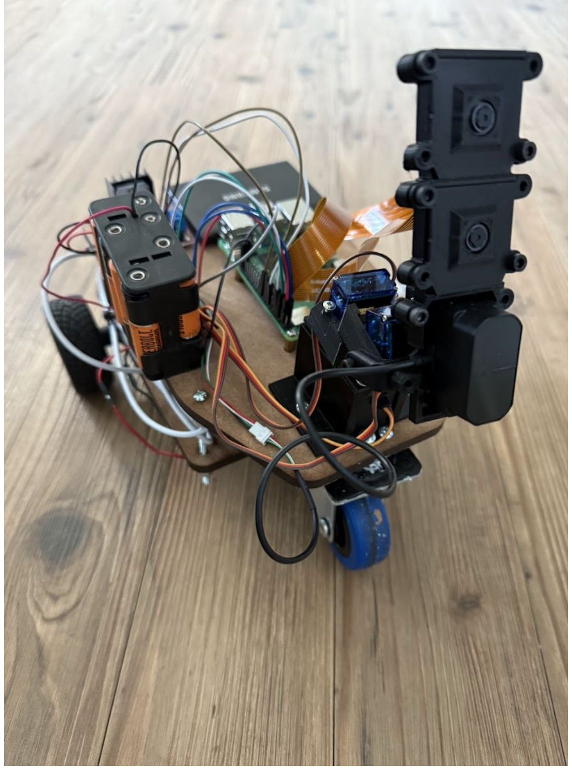
Human Radar System

Alex Do, Dhiraj Atmakuri, Arun Natarajan

Introduction

Problem Statement: Elderly and those in need of physical assistance might not have a human by their side at all times in case of an emergency fall.

Goal: Create a tracking system that follows the individual at all times, and send a message to caregiver when a fall is detected



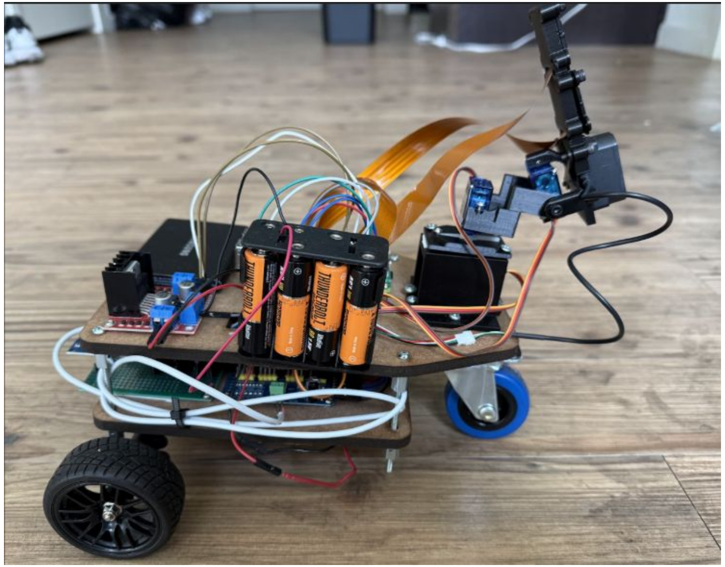
Demonstration

Physical Ware: Car Chassis

Chassis was previously constructed using two wood boards with one inches of space, and two rear wheels controlled by a Greartisan 12V DC Motor.

New Additions:

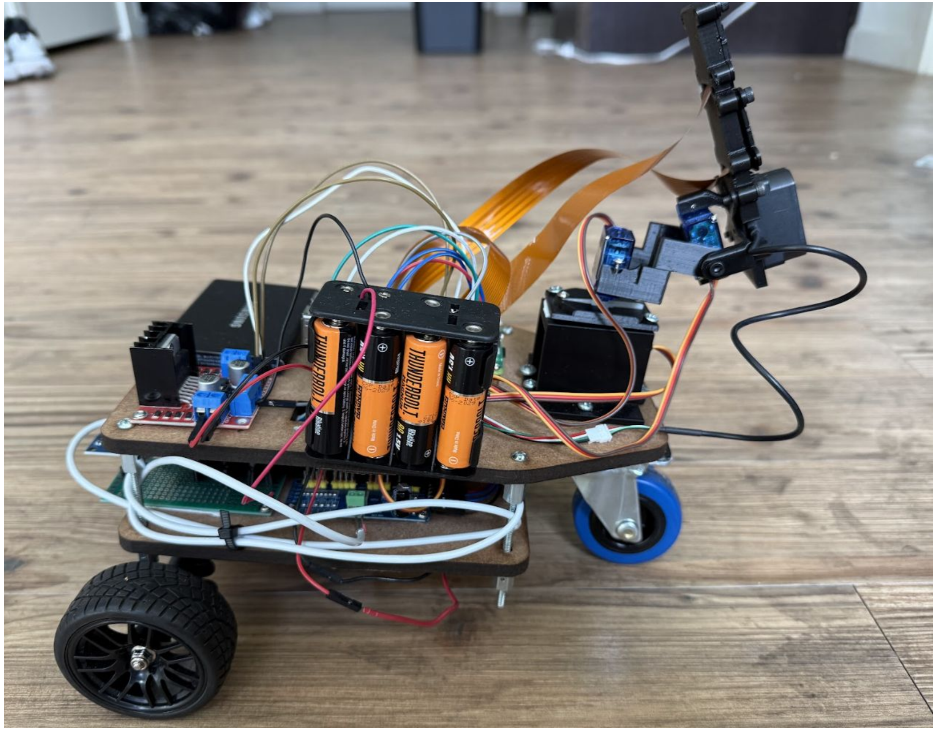
- Caster wheel, SG 5010 Servo, and Custom 3D printed housing was added to the front to give the car turning ability
- Space between boards was increased to accommodate electronic components inside the car
- Custom Housing for Cameras and LiDAR were designed and 3-D Printed



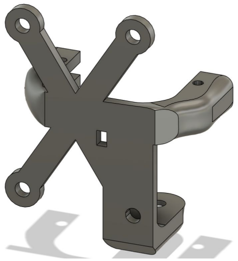
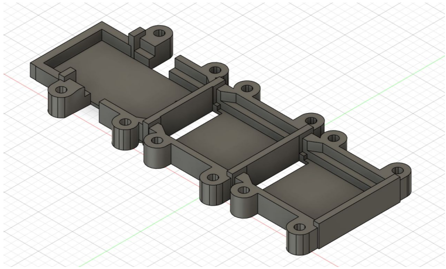
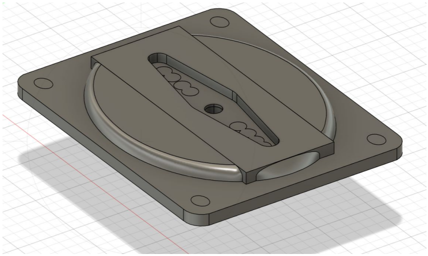
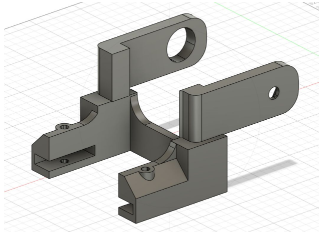
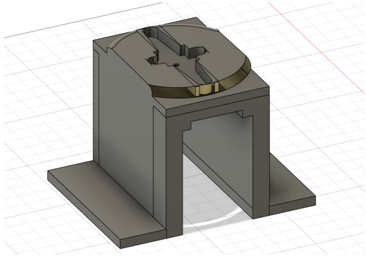
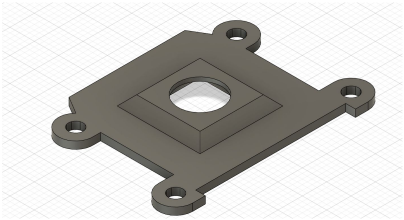
Hardware: Electronics

We utilized a variety of electronics to construct our system:

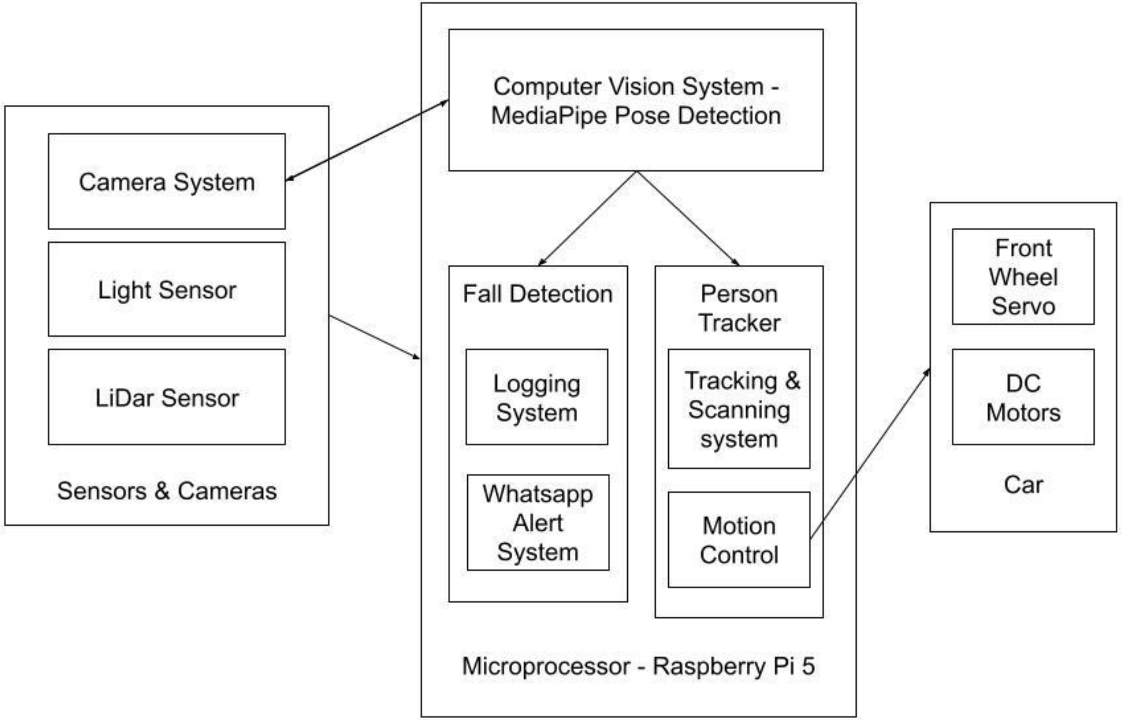
- Raspberry Pi 5
- Raspberry Pi NoIR Camera Module
- TFMini-Plus Micro LiDAR Module
- 2x SG90 Servo Motors for cameras
- 1x SG5010 Servo for front wheel
- 12V Battery via a 8AA Battery Holder
- PCA9685 PWM Controller
- 2 Battery Packs (5V/3A for Raspberry Pi, 5V/3A for PWM Controller)



Fusion360 STL Files

High Level Project Architecture



Software: Subject Tracking & Following

TEXAS A&M UNIVERSITY

Objective: Keep the subject centered in the frame and maintain an optimal following distance.

Approach:

- Uses MediaPipe Pose to identify the nose landmark and locate the subject.
- Controls the camera servos (pan and tilt) with PID control to track the subject smoothly.
- Integrates LIDAR for distance measurement:
  - Moves the car forward, backward, or stops to maintain the desired distance.

```python
def update_camera_servos(self, target_x, target_y, frame_width, frame_height):
    '''
    Uses PID control for smooth, accurate camera movements tracking the target.
    '''
    self.scanning = False
    self.person_lost_time = None

    self.last_known_position = self.current_pan

    error_x = (target_x - frame_width/2) / (frame_width/2)
    error_y = (target_y - frame_height/2) / (frame_height/2)

    self.pan_error_sum = self.pan_error_sum * 0.8 + error_x
    pan_derivative = error_x - self.last_pan_error
    pan_output = (self.Kp * error_x) + (self.Ki * self.pan_error_sum) + (self.Kd * pan_derivative)
    self.last_pan_error = error_x

    self.tilt_error_sum = self.tilt_error_sum * 0.8 + error_y
    tilt_derivative = error_y - self.last_tilt_error
    tilt_output = (self.Kp * error_y) + (self.Ki * self.tilt_error_sum) + (self.Kd * tilt_derivative)
    self.last_tilt_error = error_y

    target_pan = self.current_pan - pan_output * 60
    target_tilt = self.current_tilt - tilt_output * 60

    self.current_pan = self.current_pan * (1 - self.smooth_factor) + target_pan * self.smooth_factor
    self.current_tilt = self.current_tilt * (1 - self.smooth_factor) + target_tilt * self.smooth_factor

    self.current_pan = max(SCAN_RANGE_MIN, min(SCAN_RANGE_MAX, self.current_pan))
    self.current_tilt = max(MIN_TILT_ANGLE, min(SERVO_MAX, self.current_tilt))

    kit.servo[PAN_CHANNEL].angle = self.current_pan
    kit.servo[TILT_CHANNEL].angle = self.current_tilt

    return self.current_pan
```

```python
def control_car(self, pose_landmarks, frame_width, pan_angle, distance):
    '''
    Coordinates robot movement based on visual tracking and LIDAR data.
    '''
    steering_angle = self.get_steering_angle(pan_angle)
    kit.servo[0].angle = steering_angle

    if distance is not None:
        if distance > self.desired_distance + self.distance_tolerance:
            print(f"Moving forward - Distance: {distance:.2f}m - Steering: {steering_angle}")
            self.move_forward()
        elif distance < self.desired_distance - self.distance_tolerance:
            print(f"Moving backward - Distance: {distance:.2f}m - Steering: {steering_angle}")
            self.move_backward()
        else:
            print(f"Good distance - Distance: {distance:.2f}m - Steering: {steering_angle}")
            self.stop_motors()
    else:
        print("Distance data not available, stopping motors")
        self.stop_motors()
```

Software: Fall Detection

Objective: Detect if the tracked subject has fallen and trigger an alert.

Approach:

- Uses MediaPipe Pose to detect key landmarks (e.g., nose, hips, knees).
- Monitors the nose position relative to the frame's height.
- Checks if the nose goes below a threshold height (Y_THRESHOLD_PERCENT), indicating a potential fall.
- Confirms the fall if this condition persists for a defined duration (FALL_DELAY - 5 seconds).

```
def handle_fall_detection(self, nose_y, frame_height, current_time):
    '''
    Monitors person's vertical position and triggers alerts when falls are detected.
    '''
    y_threshold = frame_height * Y_THRESHOLD_PERCENT
    if nose_y > y_threshold:
        if not self.fall_potential:
            self.fall_potential = True
            self.fall_time = current_time
            print("Potential fall detected, starting timer...")
    else:
        self.fall_potential = False
        self.fall_time = None

    if self.fall_potential and (current_time - self.fall_time >= FALL_DELAY):
        print("Fall detected!")
        if current_time - self.last_alert_time >= self.alert_cooldown:
            if self.whatsapp_sender.send_fall_alert(self.alert_number):
                print("Fall alert sent successfully")
                self.last_alert_time = current_time
            else:
                print("Failed to send fall alert")
        else:
            print("Alert cooldown active, skipping notification")
        self.fall_potential = False
        self.fall_time = None
```

Software: Messaging

Objective: Notify a caretaker in case of a fall using a WhatsApp message.

Approach:

- Leverages the Twilio API to send WhatsApp messages.
- Retrieves approximate location via IP-based geolocation.
- Formats the message to include:
  - Alert message.
  - Location details (city, country, coordinates, etc.).
  - Google Maps link for easy navigation.

```python
def send_fall_alert(self, to_number):
    try:
        location_info = self.get_ip_location()
        full_message = self.format_location_message(location_info, "ALERT: Fall detected!")

        formatted_to = f'whatsapp:{to_number}'

        message = self.client.messages.create(
            body=full_message,
            from_=self.from_number,
            to=formatted_to
        )

        logging.info(f"Fall alert sent successfully to {to_number}")
        return True

    except Exception as e:
        error_msg = f"Failed to send fall alert to {to_number}: {str(e)}"
        logging.error(error_msg)
        print(error_msg)
        return False
```

Challenges Encountered

Chassis:

- The pre-built three-wheeled chassis lacked turning capability, severely limiting mobility. So, we engineered a custom steering solution using an SG5010 servo motor integrated with a modified caster wheel through a 3D-printed adapter for turning.

Person & Fall Detection:

- Tracking people from ground level posed unique challenges compared to wall-mounted systems. So, we utilized MediaPipe for robust tracking and developed a vertical position tracking algorithm to detect falls by monitoring if a person remains below a defined threshold for an extended period

GPS Issues:

- When our 2G GPS module proved obsolete, we implemented a software solution combining IP-based geolocation with Twilio API integration. This provides immediate messaging capabilities, ensuring reliable tracking and emergency notifications.

Follow Algorithm

- Maintaining consistent framing and following the target at an adequate distance was challenging. As a result, we implemented PID control for smooth camera movements, integrating LIDAR sensor data for precise distance maintenance

References Used

TEXAS A&M UNIVERSITY

[1] A. Rosebrock, "Pan/tilt face tracking with a Raspberry Pi and OpenCV" 2019. [Online]. Available: https://pyimagesearch.com/2019/04/01/pan-tilt-face-tracking-with-a-raspberry-pi-and-opencv/

[2] C. A. Q. Bugarin, J. M. M. Lopez, S. G. M. Pineda, Ma. F. C. Sambrano, and P. J. M. Loresco, “Machine Vision-Based Fall Detection System using MediaPipe Pose with IoT Monitoring and Alarm,” IEEE Xplore, Sep. 01, 2022. https://ieeexplore.ieee.org/document/9929527

[3]“MediaPipe Solutions guide | Edge,” Google for Developers. https://ai.google.dev/edge/mediapipe/solutions/guide

Thank You