Advanced Mechatronics Final Project: Glass-2-Bot

Utilizing Google Glass in Head-Mounted Computer Vision for the Remote Manipulation of Household Objects

Greta Perez-Haiek, Aadhav Sivakumar, Nathan Smurthwaite

May 8, 2025

Contents

0.1 Abstract

Glass-2-Bot is a combined software/hardware innovation that merges the pre-existing technology of the Google Glass Explorer Edition (2012) with Modern Computer vision and Mobile Arm Manipulator controls to manipulate and retrieve objects. A user would need to don the Google Glass, and look at the object of their choice. The Mobile Arm Manipulator will then process the head-mounted point-of-view livestream from the Google Glass using computer vision, identify the object by it's color from the robot own point-of-view, triangulate position, then move towards / pick up the specified object. The technology would be most useful for those with disabilities, as simple tasks (such as picking up dropped keys or retrieving a soda can) could be frustrating, or even dangerous to carry through. Glass-2-Bot is a solution to this problem: By combining the unique video stream generated from the Google Glass, users can (hands free) retrieve an object with minimal movement.

Chapter 1

Introduction

1.1 Problem Statement

Many people who are living at home don't have the physical ability or the energy to do basic tasks, whether it's from a disability, and injury, or old age. Tasks such as picking up a dropped object can become frustrating (or in certain cases, even dangerous) for these individuals. In these sort of situations, the individual can physically see the object that they want- however, they have no means of retrieving the object themselves. Most solutions to this pressing issue are still being developed in sterile research labs, or utilize robots that cost hundreds of thousands of dollars. The cheapest robot on the market built for household domestic tasks happens to be Hello Robot's "Stretch 3," and even then, that robot is at least 25 thousand dollars. Regardless, there is a need to develop a hands-free "fetching" solution that could alleviate the frustrations of those in need... all without breaking the bank.

1.2 The Rise and Fall of Google Glass

Google Glass was first introduced as a Moonshot Lab prototype through the 2012 San Francisco "Foundation fighting Blindness" Event- it was pitched as a revolutionary video streaming device that has the potential to change how humans interface with technology, such as aiding the disabled (particularly the blind and the physically impaired). In 2013, Google released the first edition of the Google Glass: The Explorer Edition. A limited amount was manufactured and sold as a "demo product," and soon after, Explorer Edition 2.0 and an Enterprise Edition (marketed towards working professionals) were released. Unfortunately, with the rise of unique technology means the rise of controversy, and soon, the Google Glass was met with immense criticism. The head-mounted camera highlighted concerns for privacy, and the egoistical individuals who were lucky to get a first generation Explorer Edition would proceed to be dubbed "Gl*ssholes" due to the way they aggressively filmed everything. Google Glass was quick to get themselves banned from most businesses in San Francisco, and inevitably, Google Moonshot Lab halted production in 2015.

Despite halting production, Google proceed to support Google Glass technology until 2023, where the software became officially depreciated almost a decade later. No official support means no way to connect to Google software... which means that developers could now take the reigns in how Google Glass can be used. Offloading historical or open source

APKs (a.k.a, apps) and connecting to other devices via WIFI or Bluetooth enables users to shape Glass to what they want it to be. The most useful feature of the Google Glass happens to be the unique head-mounted Point-of-view that the 5-megapixel still / 720p video camera provides, and plenty of apps take advantage of this design choice. Despite the immense controversy behind Google Glass, the potential behind the technology is limitless, especially for the original purpose of assisting the disabled.

1.3 Solution: Merging Google Glass with Mobile Arm Manipulators

How can we make useful the standing Google Glass technology for assisting the disabled? We need to create a helpful solution that allows individuals to stay in the same place and use minimal movements to obtain objects that are in sight but out of reach. Luckily, with the use of Mobile Arm Manipulators (which are wheeled robots with a dexterous robot arm with a claw on the end-effector), this can be possible.

How can we incorporate Google Glass into Mobile Arm Manipulators? By generating a head-mounted point-of-view livestream from the Google Glass into a Raspberry Pi onto the Mobile Arm Manipulator, computer vision can be conducted to locate the object of choice. Once the object is triangulated, the Raspberry Pi will serially communicate with an Arduino, which would translate the kinematics into actionable controls. The Mobile Arm Manipulator will then act upon the given controls, and the object of choice (located using the Google Glass worn by the user) will then be picked up by the robot. With Google Glass, Mobile Arm Manipulators are more enabled to help vulnerable populations and conduct tasks in a hands-free manner.

1.3.1 Introducing Glass-2-Bot

Our combined software/hardware "Glass-2-Bot" streamlines the pipeline from data collected from the eye-level point-of-view Glass Stream to manipulation policies on the Mobile Arm Manipulator that we have built. When the user focuses on an object, OpenCV will be used to detect the color of the object, and hence, it's location relative to the robot. The robot will then proceed to move forward (and turn accordingly) so that it lines up in front of the object, then uses it's claw end-effector to grab onto the object and pick it up. Since the user has to do nothing except wear the Google Glass and look at an object of choice, this object retrieval method is useful to everyone, especially for those with disabilities.

Chapter 2

Implementation and Design

2.1 Overall Approach

In order to emphasize a rapid prototyping approach of research and design, we utilized additive manufacturing for all of our physical components. As far as electrical components, the servos were chosen not only based on a continuous drive train but for actuation of a limited workspace manipulator all while keeping cost and power requirements low.

2.1.1 Hardware Development

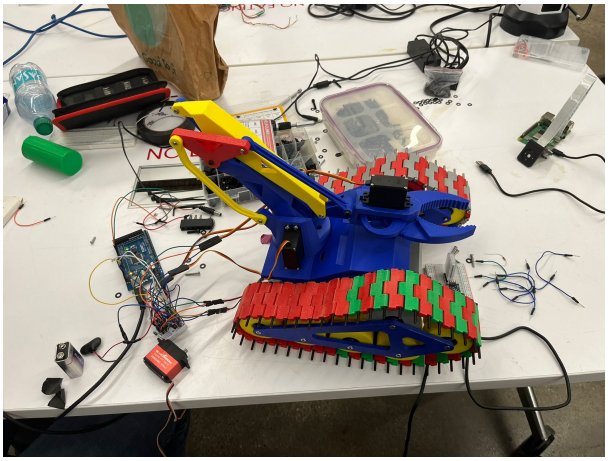


Figure 2.1: Constructed Robot

We utilized an open source mobile arm manipulator as a basis and heavily modified it and its design to serve our own niche purpose for this project and objective.

2.1.2 Software Development

We separated out tasks for software development amongst different people in order to work in parallel on different tasks necessary for the final project. One designer handled object recognition with the raspberry pi camera on board the bot, while the other handled object detection with the google glass camera module. In the end, we merged and integrated each of these tasks into 1 code file for each microcontroller.

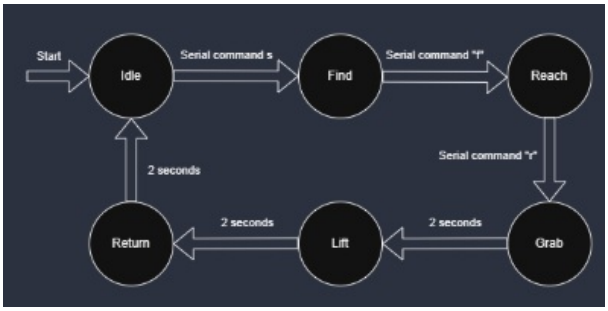


Figure 2.2: State machine

Chapter 3

Results and Conclusion

During the May 8th, 2025 demonstration, while wearing the Google Glass, a red and green object were identified successfully by the user by their colors. The robot then recognized the same objects by their color using the raspberry pi camera strapped to the front. The robot proceed to move forward and grab onto both the red and the green objects successfully. This has fulfilled our initial goal: Can a Mobile Arm Manipulator retrieve objects using a hands free approach (i.e, sight through the Google Glass)?

3.1 Future Work

3.1.1 Gripper

The conclusion of this project was met with success- however, there are plenty of things that could be improved for future iterations of the Glass-2-Bot system. For example, the gripper end-effector could be improved: During previous tests, the object occasionally fell through the gripper when trying to execute the grasping motion. When the gripper was lined with duct tape, the results were improved, leading for us to believe that the PLA that the gripper was printed with is too rigid and hence lacks the grip needed to hold onto a majority of household objects. A gripper made of more friction-prone material (such as silicon, rubber, etc.) could yield better results that prevent objects from slipping underneath it. Increasing the torque of the gripers could also help the "grasping" issue. Combing both an increased force and increased friction element would improve the project significantly.

3.1.2 Speed and Power

The robot itself was very slow when navigating towards the object at hand. While it can accomplish the assigned task outright, the speed might make it frustrating for end users to utilize. This may be due to the lack of strength in the motors chosen for the chassis. Increasing the motor torque using intelligent gear ratios, or modifying the build to accommodate for stronger motors, may be able to increase the speed, precision, and power at which the treads move.

Appendix A

Arduino code (C++)

```
#include <Servo.h>

const byte numChars = 32;
char receivedChars[numChars];   // an array to store the received data

boolean newData = false;


//IDLE 0
//searching for object 1
//moving toward object 2
//grabbing object 3
//lifting object 4
//returning 5
//reset 6
int statemachinestate=0;

int startflag=0;

int itemfoundflag=0;

int reachitemflag=0;

int grabbeditemflag=0;

int lifteditemflag=0;

int returnedflag=0;

int resetflag=0;

int targetangle=0;

Servo servo1; //gripper
Servo servo2; //arm 1
Servo servo3; //arm 2
Servo servo4; //left tank
Servo servo5; //right tank









//MPU6050 mpu;

int const INTERRUPT_PIN = 54;  // Define the interruption #0 pin
bool blinkState;

bool DMPReady = false;  // Set true if DMP init was successful
uint8_t MPUIntStatus;   // Holds actual interrupt status byte from MPU
uint8_t devStatus;      // Return status after each device operation (0 =
success, !0 = error)
uint16_t packetSize;    // Expected DMP packet size (default is 42 bytes)
uint8_t FIFOBuffer[64]; // FIFO storage buffer

float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   Yaw/Pitch/Roll container
and gravity vector

/*------Interrupt detection routine------*/
volatile bool MPUInterrupt = false;     // Indicates whether MPU6050
interrupt pin has gone high
void DMPDataReady() {
  MPUInterrupt = true;
}

int readYPRcount=0;

#define motor1Pin1  2      // IN1 on the ULN2003 driver
#define motor1Pin2  3      // IN2 on the ULN2003 driver
#define motor1Pin3  4     // IN3 on the ULN2003 driver
#define motor1Pin4  5     // IN4 on the ULN2003 driver


#define motor2Pin1  6      // IN1 on the ULN2003 driver
#define motor2Pin2  7      // IN2 on the ULN2003 driver
#define motor2Pin3  8     // IN3 on the ULN2003 driver
#define motor2Pin4  9     // IN4 on the ULN2003 driver

#define MotorInterfaceType 8

// Initialize with pin sequence IN1-IN3-IN2-IN4 for using the AccelStepper
library with 28BYJ-48 stepper motor:
```

```
void setup()
{
    servo1.attach(8);
    servo2.attach(9);
    servo3.attach(10);
    servo4.attach(11);
    servo5.attach(12);
    Serial.begin(9600);
    Serial1.begin(9600);
}

void recvWithEndMarker() {
    static byte ndx = 0;
    char endMarker = '\n';
    char rc;

    while (Serial1.available() > 0 && newData == false) {
        rc = Serial1.read();

        if (rc != endMarker) {
            receivedChars[ndx] = rc;
            ndx++;
            if (ndx >= numChars) {
                ndx = numChars - 1;
            }
        }
        else {
            receivedChars[ndx] = '\0'; // terminate the string
            ndx = 0;
            newData = true;
        }
    }
}

void showNewData() {
    if (newData == true) {
        Serial.print("This just in ... ");
        Serial.println(receivedChars);









        newData = false;
    }
}

void loop()
{

    //stepper2.run();

    switch(statemachinestate){
        case 0: //IDLE 0

            servo1.writeMicroseconds(650); // set servo to mid-point //700
open, 1500 grab cylinder, 1950 close
            servo2.writeMicroseconds(1500); // set servo to mid-point
//main arm
            servo3.writeMicroseconds(1100); // set servo to mid-point
//secondary arm
            servo4.writeMicroseconds(1500);
            servo5.writeMicroseconds(1500);
            recvWithEndMarker();
            if(receivedChars[0]==0x73){
                startflag=1;
            }
            showNewData();
            if(startflag){
                startflag=0;
                statemachinestate=1;
                Serial.println("search");
            }
            break;
        case 1: //searching for object 1
            servo4.writeMicroseconds(1750);
            servo5.writeMicroseconds(1750);
            recvWithEndMarker();
            if(receivedChars[0]==0x66){
                itemfoundflag=1;
            }
            showNewData();
            if(itemfoundflag){
```

```
            itemfoundflag=0;
            statemachinestate=2;
            Serial.println("reaching");
        }

        break;
    case 2: //reaching object 2
        servo4.writeMicroseconds(1250);
        servo5.writeMicroseconds(1750);
        recvWithEndMarker();
        if(receivedChars[0]==0x72){
            reachitemflag=1;
        }
        showNewData();
        if(reachitemflag){
            servo4.writeMicroseconds(1500);
            servo5.writeMicroseconds(1500);
            reachitemflag=0;
            statemachinestate=3;
            Serial.println("grabbing");
        }
        break;
    case 3: //grabbing object 3

        servo2.writeMicroseconds(2100); // set servo to mid-point
//main arm
        servo3.writeMicroseconds(800); // set servo to mid-point
//secondary arm
        delay(2500);
        servo1.writeMicroseconds(1700); // set servo to mid-point
        delay(2500);
        grabbeditemflag=1;
        if(grabbeditemflag){
            grabbeditemflag=0;
            statemachinestate=4;
            Serial.println("lifting");
        }
        break;
    case 4: //lifting object 4
        servo1.writeMicroseconds(1700); // set servo to mid-point









        servo2.writeMicroseconds(1500);
        servo3.writeMicroseconds(1100);
        delay(2500);
        lifteditemflag=1;
        if(lifteditemflag){
            lifteditemflag=0;
            statemachinestate=5;
            Serial.println("returning");
        }
        break;
    case 5: //returning 5
        servo4.writeMicroseconds(1750);
        servo5.writeMicroseconds(1250);
        delay(4000);
        returnedflag=1;
        if(returnedflag){
            lifteditemflag=0;
            statemachinestate=0;
            Serial.println("idle");
        }
        break;
    case 6: //reset 6

        break;
  }

}
```

Appendix B

Raspberry Pi code (Python)

```
import requests
import serial
import cv2
import numpy as np
import time
from picamera2 import Picamera2

SERIAL_PORT = '/dev/serial0'
BAUD_RATE = 9600
MESSAGE_INTERVAL = 0.1
ser = None

ser = serial.Serial(SERIAL_PORT, BAUD_RATE)
time.sleep(0.1)


PREVIEW_WIDTH = 640
PREVIEW_HEIGHT = 480

greenglasscenterx=0
greenglasscentey=0
yellowglasscenterx=0
yellowglasscentery=0
redglasscenterx=0
redglasscentery=0

reddetectcount=0
yellowdetectcount=0
greendetectcount=0

glasscenterx=160
glasscentery=120

DetectedObject="None"

picamcenterobject="None"

ArduinoState=0

lower_y_bound = np.array([22, 90, 100])
upper_y_bound = np.array([38, 255, 255])
lower_r_bound = np.array([110, 100, 70])
upper_r_bound = np.array([180, 255, 255])
lower_g_bound = np.array([40, 50, 40])









upper_g_bound = np.array([70, 255, 255])
MIN_CONTOUR_AREA_COLOR = 400

lower_color_bound = np.array([0, 0, 0])
upper_color_bound = np.array([0, 0, 0])

CENTER_REGION_PERCENT_W = 0.05
CENTER_REGION_PERCENT_H = 1.0
center_x = PREVIEW_WIDTH // 2
center_y = PREVIEW_HEIGHT // 2

center_region_half_w = int((PREVIEW_WIDTH * CENTER_REGION_PERCENT_W) / 2)
center_region_half_h = int((PREVIEW_HEIGHT * CENTER_REGION_PERCENT_H) / 2)

center_x_min = center_x - center_region_half_w
center_x_max = center_x + center_region_half_w
center_y_min = center_y - center_region_half_h
center_y_max = center_y + center_region_half_h

picam2 = None

picam2 = Picamera2()
preview_config = picam2.create_preview_configuration(main={"size": (PREVIEW_WIDTH,
PREVIEW_HEIGHT), "format": "RGB888"})
picam2.configure(preview_config)
picam2.start()
time.sleep(0.1)
cv2.namedWindow("Object in Center Detection", cv2.WINDOW_AUTOSIZE)
MJPEG_STREAM_URL = "http://10.253.210.57:8080/cam.mjpeg"
REQUEST_TIMEOUT = 10
MIN_CONTOUR_AREA = 80


kernel = np.ones((5, 5), np.uint8)


WINDOW_ORIGINAL = 'Live Stream'
WINDOW_MASK = 'Yellow Mask'
WINDOW_DETECTED = 'Detected Yellow Objects'

def detect_object(in_frame,lower_hsv_bound,upper_hsv_bound):
    x, y, w, h =0, 0, 0, 0

    hsv_frame = cv2.cvtColor(in_frame, cv2.COLOR_BGR2HSV)
```

```
    mask = cv2.inRange(hsv_frame, lower_hsv_bound, upper_hsv_bound)

    mask_cleaned = cv2.erode(mask, kernel, iterations=1)
    mask_cleaned = cv2.dilate(mask_cleaned, kernel, iterations=1)

    contours, _ = cv2.findContours(mask_cleaned, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    detections_found_this_frame = 0
    for contour in contours:
        area = cv2.contourArea(contour)
        if area > MIN_CONTOUR_AREA:
            detections_found_this_frame += 1
            x, y, w, h = cv2.boundingRect(contour)

    return x, y, w, h

def detect_all_objects(raw_cam_frame):
    global reddetectcount
    global greendetectcount
    global yellowdetectcount
    global DetectedObject

    output_frame = raw_cam_frame.copy()
    xyel, yyel, wyel, hyel=detect_object(raw_cam_frame, np.array([20, 100, 100]), np.array([30,
255, 255])) #yellow
    xgre, ygre, wgre, hgre = detect_object(raw_cam_frame, np.array([40, 50, 40]), np.array([85,
255, 255])) # green
    xred, yred, wred, hred = detect_object(raw_cam_frame, np.array([130, 40, 30]),
np.array([200, 255, 255]))  # red


    cv2.rectangle(output_frame, (xyel, yyel), (xyel + wyel, yyel + hyel), (0, 255, 255), 2)
    cv2.putText(output_frame, 'Yellow', (xyel, yyel - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
255, 255), 2)
    yellowglasscenterx=xyel+wyel/2
    yellowglasscentery=yyel+hyel/2

    if(yellowglasscenterx>130 and yellowglasscenterx<190 and yellowglasscentery<150 and
yellowglasscentery>90):
        yellowdetectcount+=1
        if(yellowdetectcount>20):
            DetectedObject="yellow"









            print(DetectedObject)
    else:
        yellowdetectcount=0


    cv2.rectangle(output_frame, (xgre, ygre), (xgre + wgre, ygre + hgre), (0, 255, 0), 2)
    cv2.putText(output_frame, 'Green', (xgre, ygre - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
255, 0), 2)
    greenglasscenterx=xgre+wgre/2
    greenglasscentery=ygre+hgre/2

    if(greenglasscenterx>130 and greenglasscenterx<190 and greenglasscentery<150 and
greenglasscentery>90):
        greendetectcount+=1

        if(greendetectcount>20):
            DetectedObject="green"
            print(DetectedObject)
    else:
        greendetectcount=0

    cv2.rectangle(output_frame, (xred, yred), (xred + wred, yred + hred), (0, 0, 255), 2)
    cv2.putText(output_frame, 'Red', (xred, yred - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0,
255), 2)
    redglasscenterx=xred+wred/2
    redglasscentery=yred+hred/2

    if(redglasscenterx>130 and redglasscenterx<190 and redglasscentery<150 and
redglasscentery>90):
        reddetectcount+=1
        if(reddetectcount>20):
            DetectedObject="red"
            print(DetectedObject)
    else:
        reddetectcount=0

    return output_frame

def process_and_detect(stream_url):
    global ArduinoState
    global picamcenterobject
    global lower_color_bound
    global upper_color_bound
    global DetectedObject
```

```
    bytes_buffer = b''
    last_frame_time = time.time()
    frames_processed = 0

    try:

        print(f"Attempting to connect to stream: {stream_url}")
        response = requests.get(
            stream_url,
            stream=True,
            timeout=REQUEST_TIMEOUT
        )
        response.raise_for_status()
        print("Connection successful. Reading stream...")
        print("Press 'q' in one of the OpenCV windows to quit.")

        cv2.namedWindow(WINDOW_DETECTED, cv2.WINDOW_NORMAL)

        for chunk in response.iter_content(chunk_size=4096): # Increased chunk size slightly
            if not chunk:
                print("Stream ended or chunk was empty.")
                break

            bytes_buffer += chunk
            start_marker = b'\xff\xd8'
            end_marker = b'\xff\xd9'

            a = bytes_buffer.find(start_marker)
            b = bytes_buffer.find(end_marker)

            if a != -1 and b != -1 and b > a:
                jpg_data = bytes_buffer[a : b + 2]
                bytes_buffer = bytes_buffer[b + 2 :]


                frame = cv2.imdecode(np.frombuffer(jpg_data, dtype=np.uint8),
cv2.IMREAD_COLOR)

                if frame is not None:
                    frames_processed += 1

                    frame_rgb = picam2.capture_array("main")









                    cv2.rectangle(frame_rgb,(center_x_min, center_y_min),(center_x_max,
center_y_max),(255, 0, 0), 2)

                    hsv_frame = cv2.cvtColor(frame_rgb, cv2.COLOR_RGB2HSV)
                    color_mask = cv2.inRange(hsv_frame, lower_color_bound, upper_color_bound)

                    kernel = np.ones((5,5),np.uint8)
                    mask_processed = cv2.morphologyEx(color_mask, cv2.MORPH_OPEN, kernel)
                    mask_processed = cv2.morphologyEx(mask_processed, cv2.MORPH_CLOSE,
kernel)

                    contours, _ = cv2.findContours(mask_processed.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

                    object_in_center = False

                    if contours:
                        for contour in contours:
                            area = cv2.contourArea(contour)
                            if area > MIN_CONTOUR_AREA_COLOR:
                                M = cv2.moments(contour)
                                cX = -1
                                cY = -1
                                if M["m00"] != 0:
                                    cX = int(M["m10"] / M["m00"])
                                    cY = int(M["m01"] / M["m00"])
                                x, y, w, h = cv2.boundingRect(contour)

                                object_color = (0,0, 0)
                                status_text = "Detected"

                                if cX != -1 and (center_x_min < cX < center_x_max) and \
                                   (center_y_min < cY < center_y_max):
                                    object_in_center = True
                                    object_color = (0, 255, 0)
                                    status_text = "IN CENTER"
                                    cv2.putText(frame_rgb, status_text, (center_x - 50, center_y -
center_region_half_h - 10),
                                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, object_color, 2)


                                cv2.drawContours(frame_rgb, [contour], -1, object_color, 2)
                                cv2.rectangle(frame_rgb, (x, y), (x+w, y+h), object_color, 2)
                                if cX != -1: # Draw centroid
```

```
                        cv2.circle(frame_rgb, (cX, cY), 5, object_color, -1)
                        cv2.putText(frame_rgb, f"Obj ({status_text})", (x, y-10),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, object_color, 1)

                if not contours or not object_in_center :
                    is_object_detected_outside_center = any(cv2.contourArea(c) >
MIN_CONTOUR_AREA_COLOR for c in contours)
                    if not is_object_detected_outside_center and not object_in_center:
                        cv2.putText(frame_rgb, "Searching...", (20, 30),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255,255,255), 2)

                cv2.imshow("Object in Center Detection", frame_rgb)

                disp_frame=detect_all_objects(frame)

                cv2.imshow(WINDOW_DETECTED, disp_frame) # Show frame with detections
                if(DetectedObject!="None" and ArduinoState==0):
                    message = f"s\n"
                    ser.write(message.encode('utf-8'))
                    time.sleep(MESSAGE_INTERVAL)
                    print("SENT1")
                    ArduinoState=1
                    if(DetectedObject=="green"):
                        lower_color_bound = lower_g_bound
                        upper_color_bound = upper_g_bound
                    elif(DetectedObject=="yellow"):
                        lower_color_bound = lower_y_bound
                        upper_color_bound = upper_y_bound
                    elif(DetectedObject=="red"):
                        lower_color_bound = lower_r_bound
                        upper_color_bound = upper_r_bound
                if(ArduinoState==1 and object_in_center):
                    message = f"f\n"
                    ser.write(message.encode('utf-8'))
                    time.sleep(MESSAGE_INTERVAL)
                    print("SENT2")
                    ArduinoState=2
                if(ArduinoState==2 and w>285):
                    message = f"r\n"
                    ser.write(message.encode('utf-8'))
                    time.sleep(MESSAGE_INTERVAL)
                    print("SENT3")
                    DetectedObject="None"
                    ArduinoState=0









                if cv2.waitKey(1) & 0xFF == ord('q'):
                    print("Exit key pressed.")
                    break

                current_time = time.time()
                if current_time - last_frame_time >= 1.0:
                    fps = frames_processed / (current_time - last_frame_time)
                    print(f"Approx FPS: {fps:.2f}")
                    last_frame_time = current_time
                    frames_processed = 0
            else:
                print("Warning: Failed to decode frame.")

    except requests.exceptions.Timeout:
        print(f"Error: Connection timed out after {REQUEST_TIMEOUT} seconds.")
    except requests.exceptions.RequestException as e:
        print(f"Error connecting to or reading from stream: {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        import traceback
        traceback.print_exc()
    finally:
        print("Closing resources...")
        cv2.destroyAllWindows()
        print("Stream processing finished.")
if __name__ == "__main__":
    if MJPEG_STREAM_URL == 'YOUR_MJPEG_STREAM_URL_HERE':
        print("\nError: Please replace 'YOUR_MJPEG_STREAM_URL_HERE' with your actual
stream URL in the script.\n")
    else:
        process_and_detect(MJPEG_STREAM_URL)
```

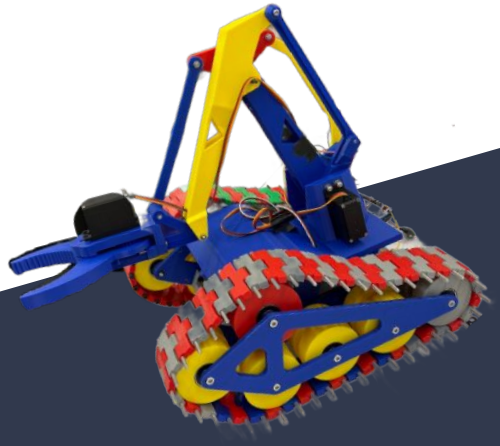
Appendix C

Final Presentation Slideshow

Glass-2-Bot

Head-Mounted Computer Vision for the Remote Manipulation of Household Objects

Team 1: Aadhav Sivakumar, Greta Perez-Haiek, Nathan Smurthwaite



Advanced Mechatronics Final Presentation || Spring 2025

Background

The Problem

Many people who are living at home don't have the physical ability or the energy to do basic tasks, whether it's from a *disability*, an *injury*, or *old age*.

They can physically **see the object they want**, but wish they have the means to retrieve it.

The Rise and Fall of Google Glass.

- Google Moonshot Lab's launch @ 2012
- Fizzled out Production @ 2015 (a controversial technology)
  - Privacy concerns (and lots of lawsuits)
- Officially Deprecated as of 2023
- No more official support = developer's playground
  - Offloaded APKs (apps)
  - Most useful feature: **Video Streaming**
  - **Incredible potential for unique applications…**

The Solution...

1) How can we integrate (and make useful) Google Glass technology for Mobile Arm Manipulators?



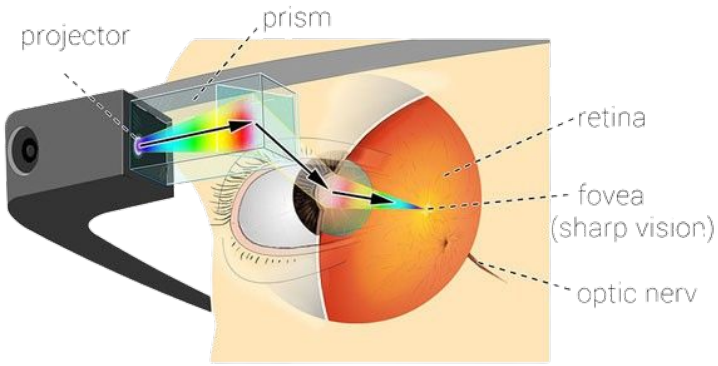

   - Answer: **Live stream → OpenCV → MoMa Robotic Controls**

2) How can Mobile Arm Manipulators help vulnerable populations (such as those with disabilities, or the elderly?)



   - Answer: **MoMa Robotic Controls → Object manipulation!**

Object Recognition with a 1st Person POV.



- Google Glass enables a unique, head-mounted POV
  - Livestreams a video right at the eyelevel from the GG to the raspberry pi
  - This livestream can be processed for object recognition purposes.
  - **If a person sees an object they want, the Pi will identify it by its color.**

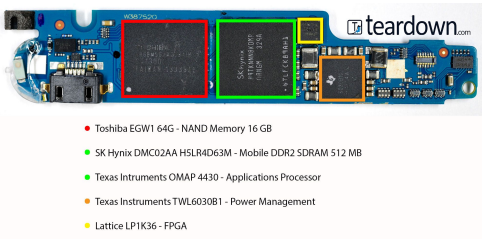
Object Recognition with a 1st Person POV.

If you wore the Google Glass with Glass-2-Bot running, this will be what you see.

Technical Methodology

3 Microcontroller Com.

TI OMAP4430
System-on-a-Chip
- Dual-Core Processor with an ***ARM Cortex-A9*** architecture.



Wi-Fi

Raspberry Pi Model 4B, 2Gb
- ***Quad core Cortex-A72 (ARM v8)***

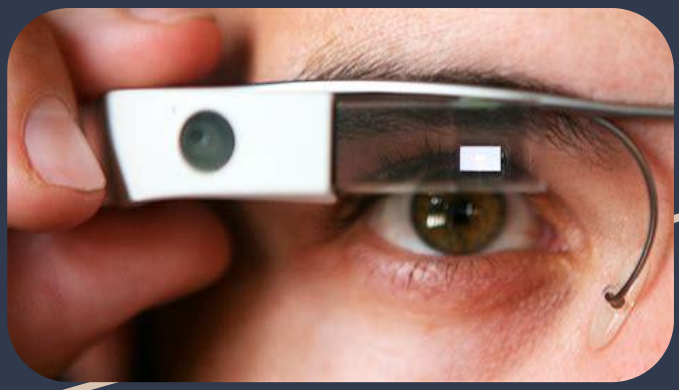
UART

Arduino Mega 2560
- ***AVR 8-bit architecture***

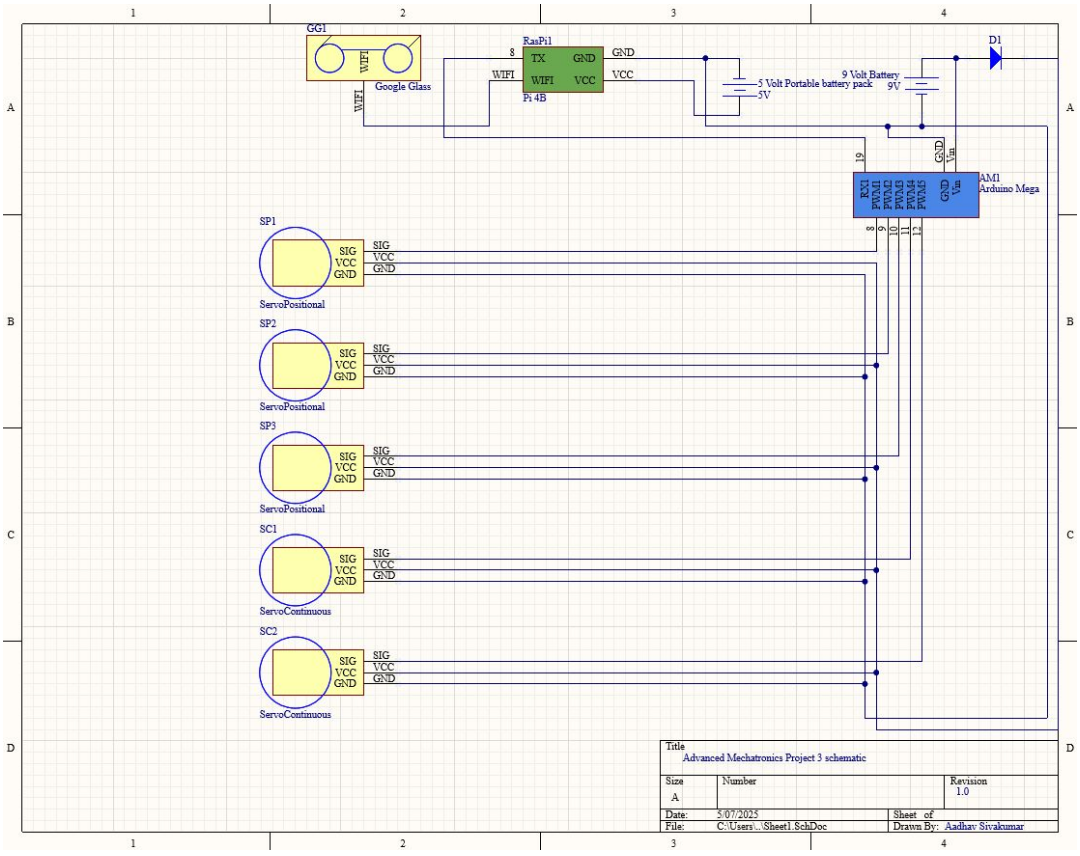
Electrical Components

- 1 Rpi Cam V2.1
- 2 *Cont.* Servo motors
- 3 *Pos.* Servo motors
- Diodes
- 9V Battery
- LiPo 5V Battery Pack

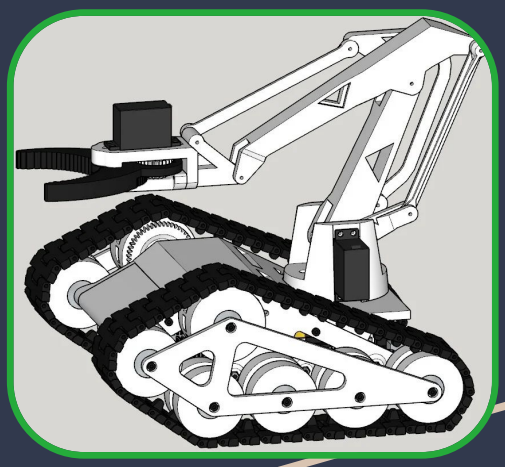
Google Glass Specs...

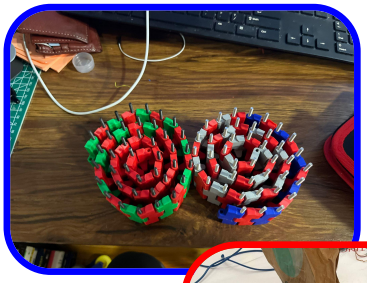


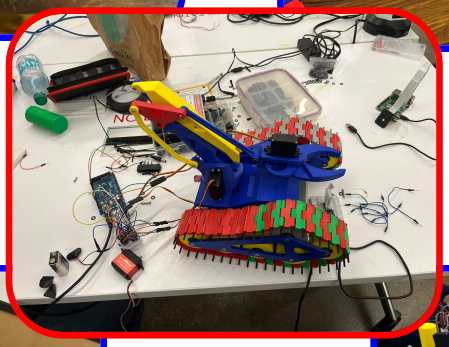
- We are using the Google Glass Explorer Edition (2013)
  - 5 megapixel still/720p video camera
  - Wifi/Bluetooth Capabilities
  - Launched during the 2012 Foundation Fighting Blindness event in San Francisco
    - GG Design roots in POV video recording
    - Theorized that it could potentially help the disabled

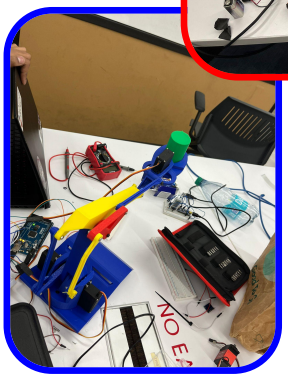


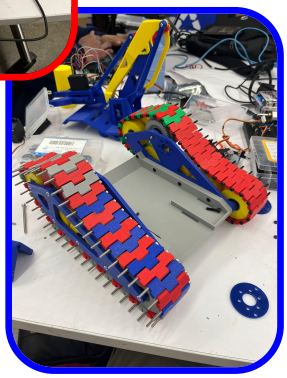
Mechanical Progress





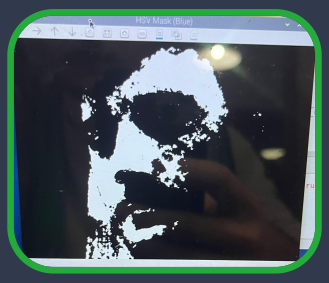


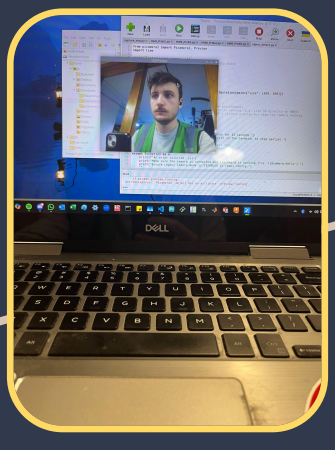


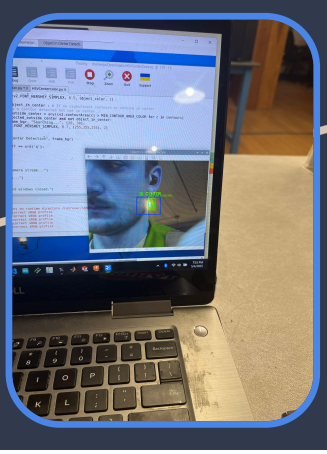


Computer Vision Progress

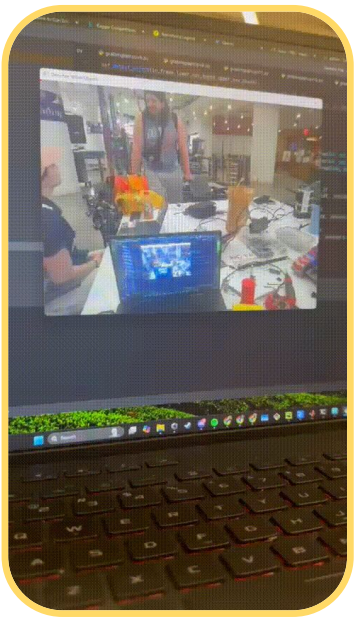
Pi Cam V2.1

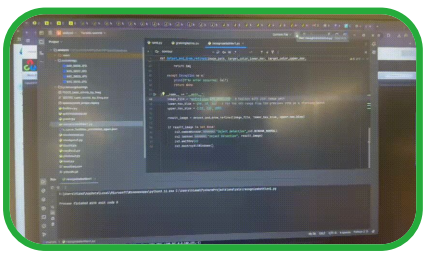




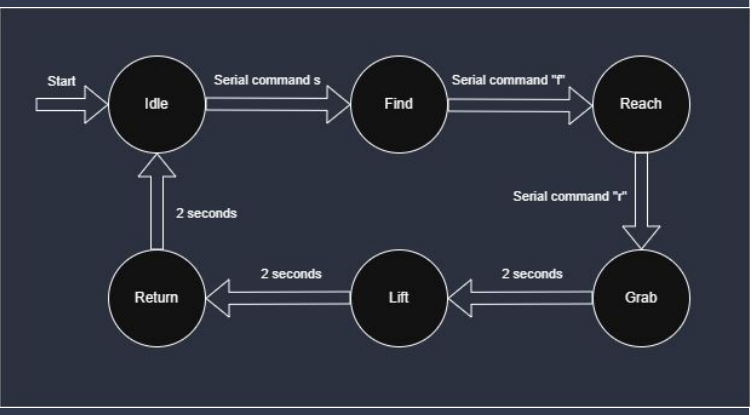


Google Glass Camera Module





Programming Flow



- **Google Glass Video Capture**
- **Raspberry Pi Image Processing**
  - OpenCV
    - RGB (Red Green Blue)
    - HSV (Hue Saturation Value)
    - Contour
- **Arduino**
  - Continuous Servos
    - Turn Bot
    - Move Forward
    - Move Back
  - Positional Servos
    - Reach down
    - Grab
    - Bring Up

Demonstrations

Demo Video #1) Red Object Detection & Manipulation

Demo Video #2) Green Object Detection & Manipulation

Demo Video #3) Red & Green Multi-Object Detection & Manipulation