Camera Tracking on Moving Objects using Raspberry Pi

Our progress so far

Let’s look at our project goals again…

Camera Tracker

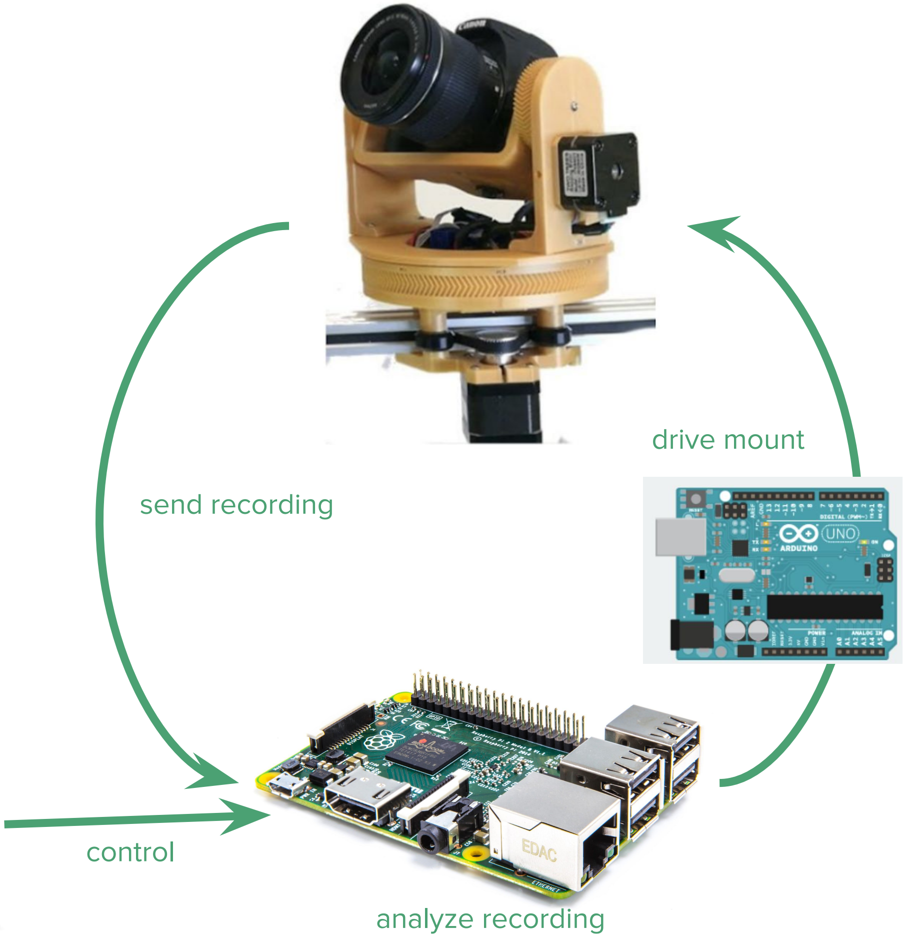
- Inspiration: [A stepper motor driven, 3D printed and Arduino controlled pan/tilt mount.]
    - Daniel Richter provides most parts
    - Pan-Tilt-Mount is controlled via Xbox controller

Drive mount using wireless joystick input

Camera Tracker

- Spice things up slightly by:
  - Using Raspberry Pi (OpenCV support) as controller
  - Implementing a simple tracking algorithm
  - Limit to one axis

→ Improve appropriately (e.g. more axes, advanced tracking algorithms vs. faster tracking)

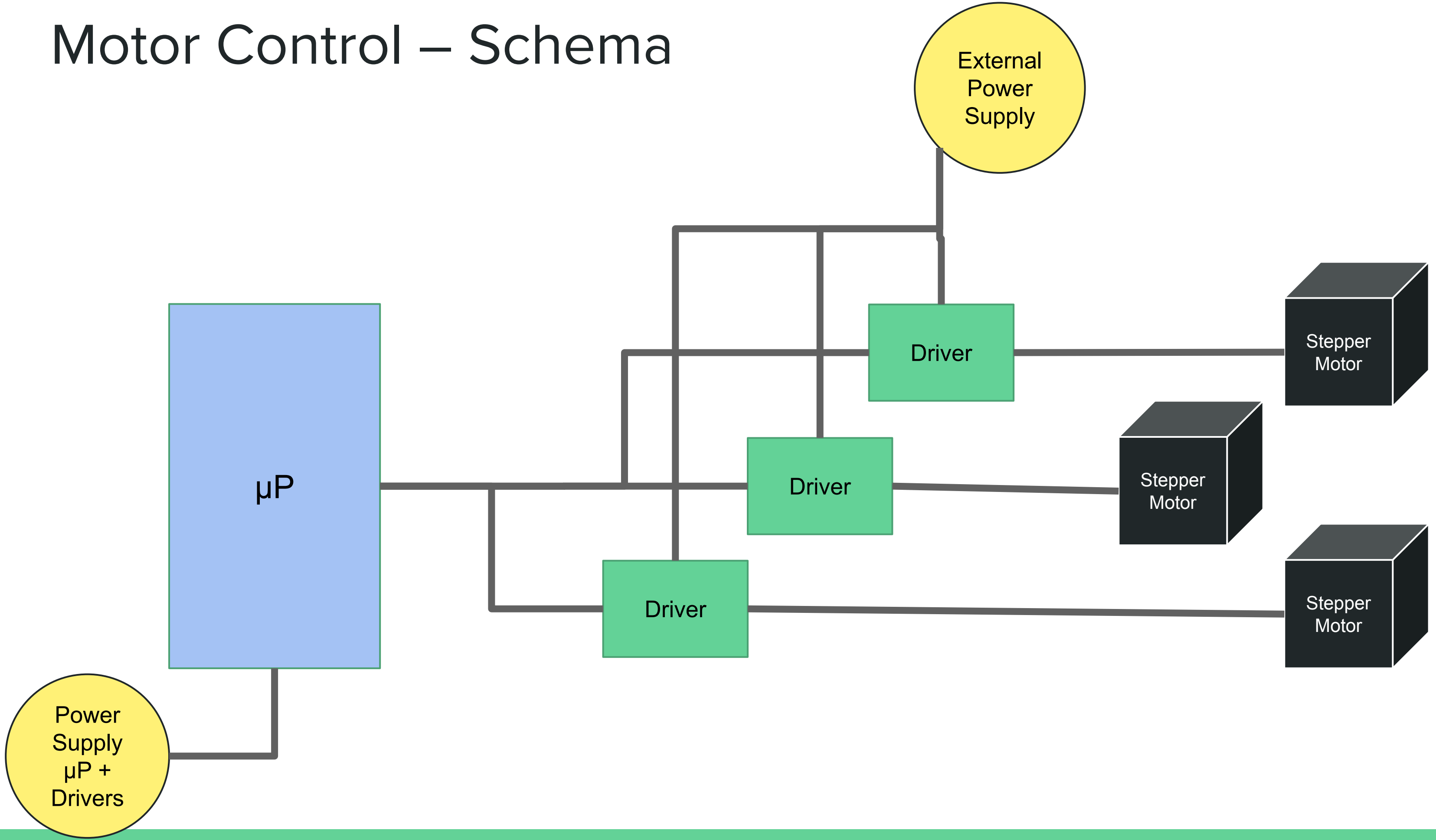


How far have we come in achieving these goals?

Two subgroups:

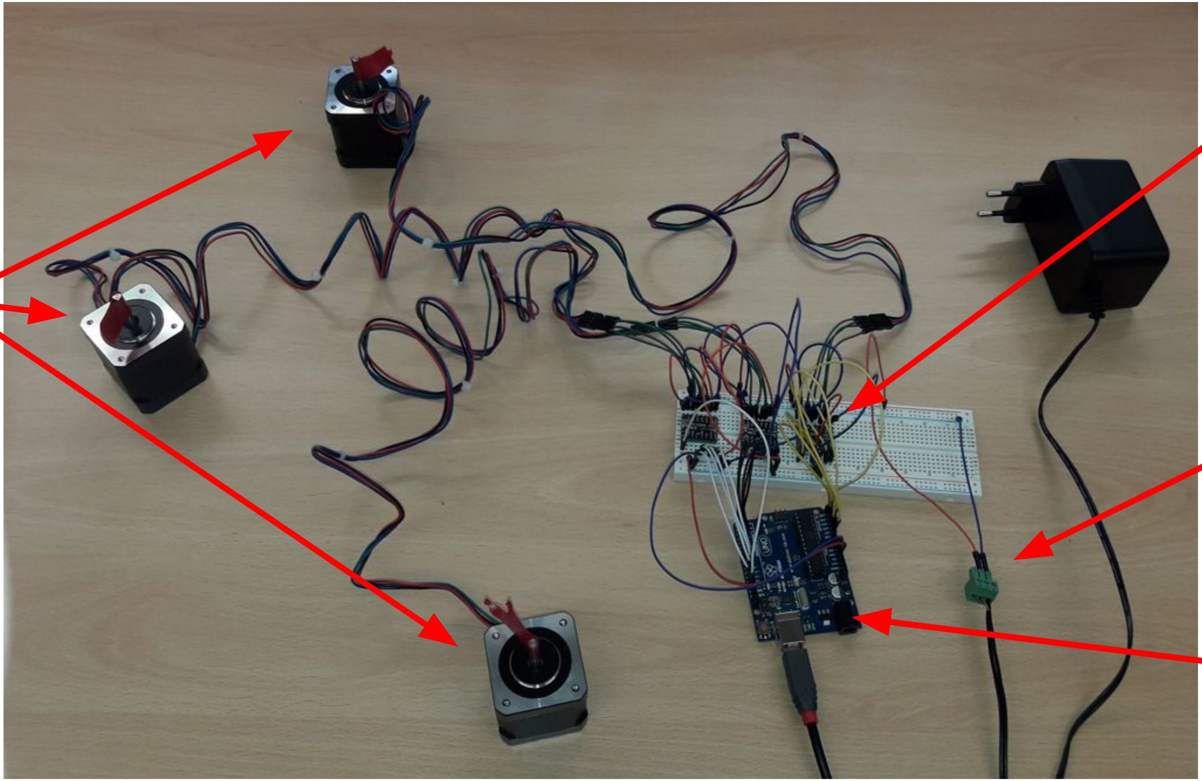
1. Motor Control
2. Object Recognition

Motor Control – Schema



Motor Control – Reality

Nema 17
Stepper
Motors



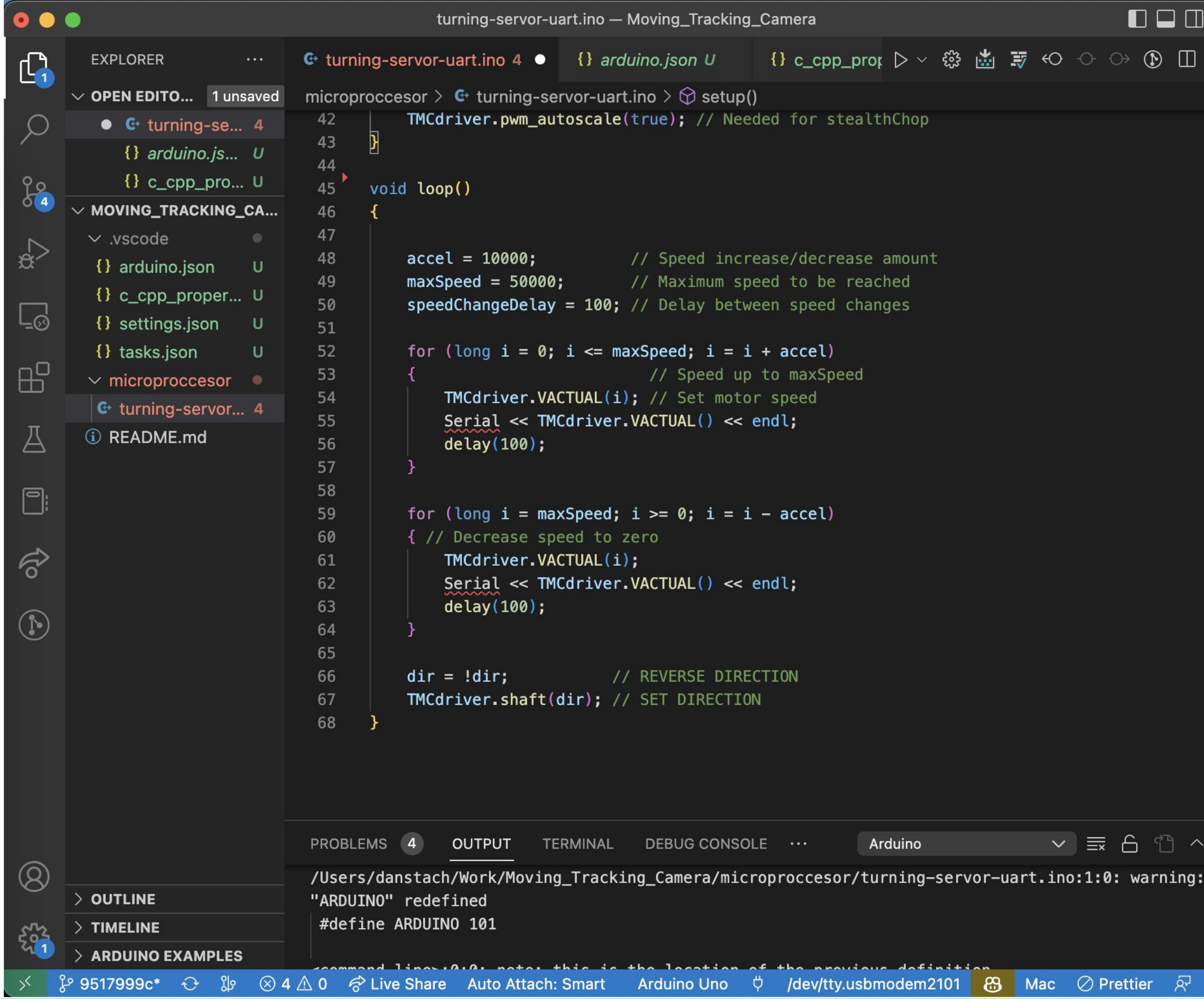
Stepper Drivers
(TMC2209)

External
Power
Supply
(12V/1A)

Arduino
Uno

Hello Stepper

- IDE: VSCode (has Arduino support)
- Use TMCStepper library to control driver
- UART interface can set motor velocity
- Use SoftwareSerial library to fake serial interface over digital pins



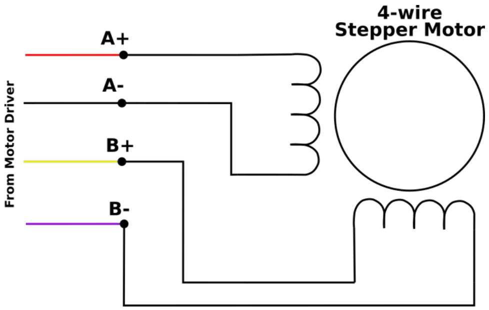
Hello Steppers

- Use AccelStepper library (most commonly used library for steppers with Arduino)
- AccelStepper uses DIRECTION and STEP pins for control. TMCStepper uses UART interface
- TMC2209 driver supports both operation modes
- .run() ist non-blocking and may do nothing, but must be called often

```
#include <TMCStepper.h>
#include <SoftwareSerial.h>
#include <Streaming.h>
#include <AccelStepper.h>

/** ... */

void setup()
{
    for (int i = 0; i < sizeMotors; i++)
    {
        SoftSerials[i].begin(11520);       // initialize software serial for UART motor control
        TMCdrivers[i].beginSerial(11520); // Initialize UART
        TMCdrivers[i].begin();             // UART: Init SW UART with default 115200 baudrate
        TMCdrivers[i].toff(5);             // Enables driver in software
        TMCdrivers[i].rms_current(400);    // Set motor RMS current
        TMCdrivers[i].microsteps(16);      // Set microsteps

        TMCdrivers[i].en_spreadCycle(false);
        TMCdrivers[i].pwm_autoscale(true); // Needed for stealthChop

        steppers[i].setMaxSpeed(800 * steps_per_mm);      // 100mm/s @ 80 steps/mm
        steppers[i].setAcceleration(1000 * steps_per_mm); // 2000mm/s^2
        steppers[i].setEnablePin(EN_PIN + MOTOR_PIN_OFFSETS[i]);
        steppers[i].setPinsInverted(false, false, true);
        steppers[i].enableOutputs();
    }
}

void loop()
{
    for (int i = 0; i < sizeMotors; i++)
    {
        if (steppers[i].distanceToGo() == 0)
        {
            steppers[i].move(100 * steps_per_mm); // Move 100mm
        }
        steppers[i].run();
    }
}
```

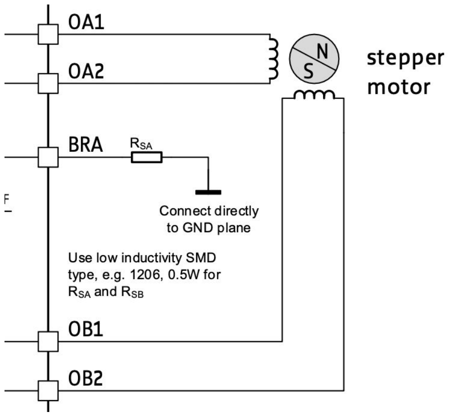
Struggle 1: Correctly Connecting Motor to Driver

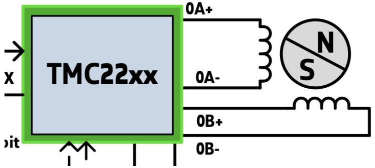
Usually:



Source: How to Wire Stepper Motors

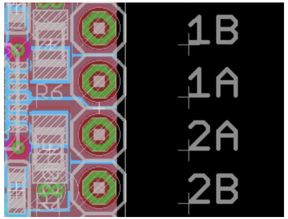
TMC Manual:





Source: TMC2209-LA - Trinamic

MKS Driver:



Source: MKS-StepStick-Driver: TMC2209

Source: [1]

Struggle 2: Setting Reference Voltage (V_{REF})

- Driver has small potentiometer to control max motor current

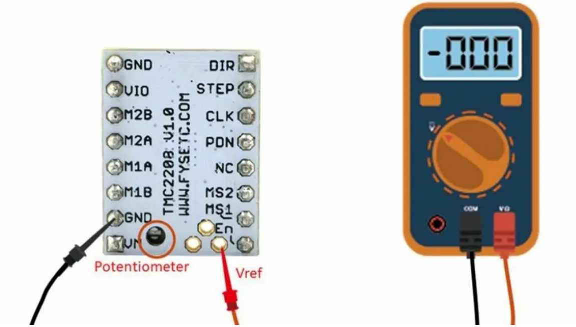
- Measure voltage with multimeter

- Calculate using

$$I_{RMS} = \frac{325\ mV}{R_{sense} + 20\ m\Omega} \cdot \frac{1}{\sqrt{2}} \cdot \frac{V_{ref}}{2.5\ V}$$

- … profit? 💸

Source: [1]



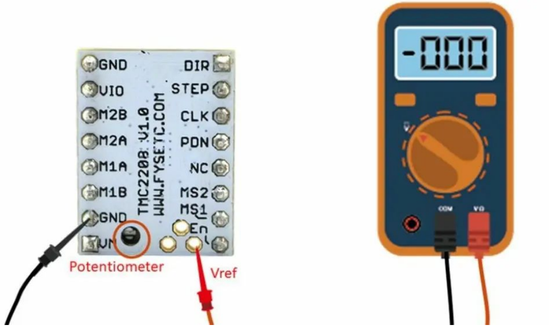
Source: Vref Calculator: How to Tune Your Stepper Driver | All3DP

Struggle 2: Setting Reference Voltage (V_{REF})

Source: [1]

- Driver has small potentiometer to control max motor current
- Measure voltage with multimeter
- Calculate using

$$I_{RMS} = \frac{325\ mV}{R_{sense} + 20\ m\Omega} \cdot \frac{1}{\sqrt{2}} \cdot \frac{V_{ref}}{2.5\ V}$$



Source: Vref Calculator: How to Tune Your Stepper Driver | All3DP

- Measurements were never in valid range! 😣
- First driver's potentiometer didn't work

Struggle 2: Setting Reference Voltage (V_{REF})

- Driver has small potentiometer to control max motor current

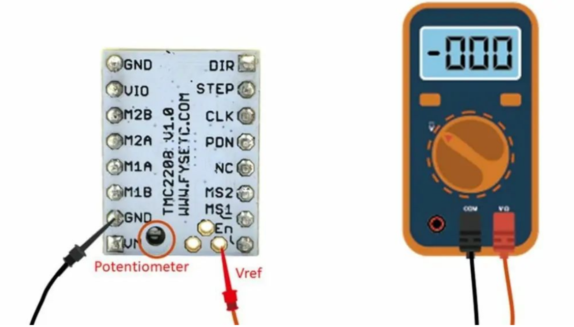
- Measure voltage with multimeter

- Calculate using

$$I_{RMS} = \frac{325\,mV}{R_{sense} + 20\,m\Omega} \cdot \frac{1}{\sqrt{2}} \cdot \frac{V_{ref}}{2.5\,V}$$

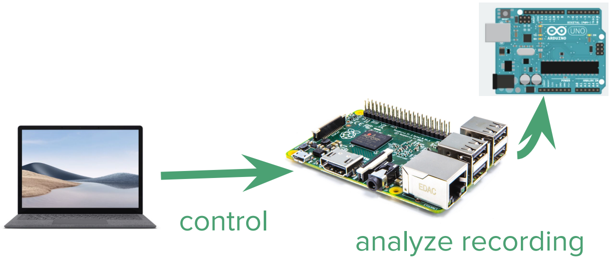
- ➡ Max current can be set using UART alternatively 🤞

Source: [1]

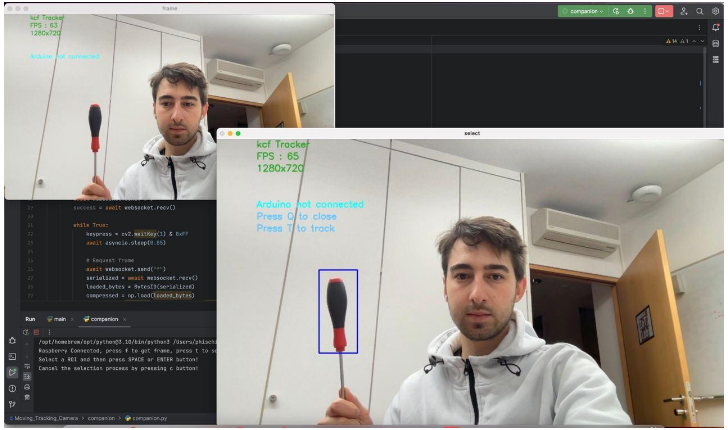


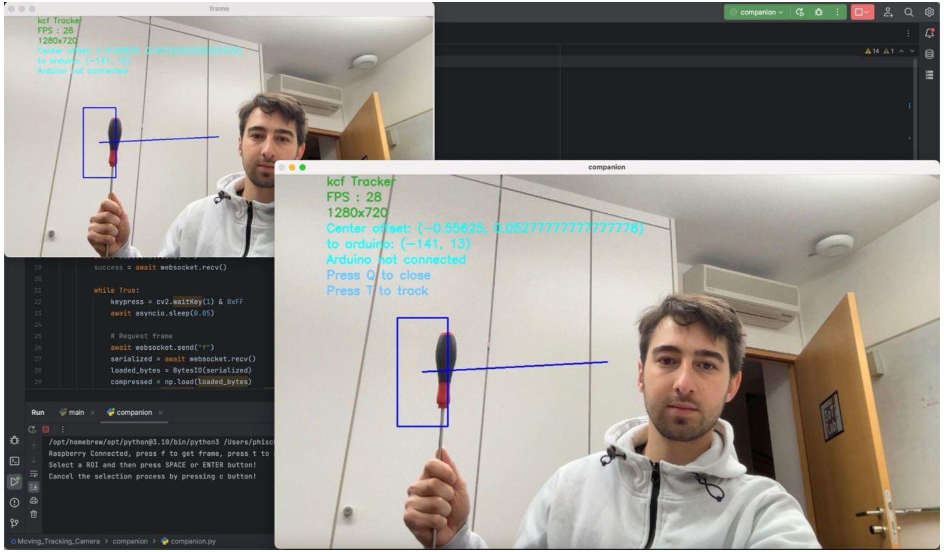
Source: Vref Calculator: How to Tune Your Stepper Driver | All3DP

Object Recognition using Raspberry Pi



- object recognition and websocket server in Python
  - OpenCV for video capture and object tracking
  - PySerial for connection to Arduino
  - Websocket server for companion app
- companion app
  - To select the object to be tracked





Implementation

```python
async def client():
    async with websockets.connect("ws://" + SERVER + ":8764", max_size=None, read_limit=2 ** 20) as websocket:
        print("Raspberry Connected, press f to get frame, press t to select tracking area")

        # Handshake
        await websocket.send("c")
        success = await websocket.recv()

        while True:
            keypress = cv2.waitKey(1) & 0xFF
            await asyncio.sleep(0.05)

            # Request frame
            await websocket.send("f")
            serialized = await websocket.recv()
            loaded_bytes = BytesIO(serialized)
            compressed = np.load(loaded_bytes)
            frame = cv2.imdecode(compressed, cv2.IMREAD_UNCHANGED)

            # Alter and show frame
            cv2.putText(frame, "Press Q to close", (100, 200), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (255, 200, 100), 2)
            cv2.putText(frame, "Press T to track", (100, 230), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (255, 200, 100), 2)
            cv2.imshow('companion', frame)

            if keypress == ord('t'):
                cv2.destroyWindow('companion')
                bbox = cv2.selectROI('select', frame, False)
                cv2.imshow('companion', frame)
                cv2.destroyWindow('select')

                # Request rpi to track image region in bbox
                await websocket.send("t")
                await websocket.send(str(bbox))
            elif keypress == ord('q'):
                cv2.destroyAllWindows()
                break


asyncio.run(client())
```

Phisch

```python
async def handler(websocket):
    global STATE, LAST_SEND_FRAME, TO_TRACK, ACTIVE_FRAME
    async for message in websocket:
        print(message)

        if message == "c":
            print("Companion is connected")
            await websocket.send("ok")
        elif message == "f":
            print("Frame requested")
            LAST_SEND_FRAME = ACTIVE_FRAME

            # Compress frame
            ok, compressed = cv2.imencode('.jpg', LAST_SEND_FRAME)

            # Save to Byte Stream
            np_bytes = BytesIO()
            np.save(np_bytes, compressed, allow_pickle=True)

            await websocket.send(np_bytes.getvalue())
        elif message == "t":
            print("Track region received")
            ROI = eval(await websocket.recv())
            TO_TRACK = (LAST_SEND_FRAME, ROI)
            STATE = "TRACK"
```

Implementation

```python
# Main Loop
while True:
    # Capture frame-by-frame
    ret, frame = video.read()
    if type(frame) == type(None):
        print("!!! Couldn't read frame!")
        break

    f_height = len(frame)
    f_width = len(frame[0])
    f_center = (int(f_width / 2), int(f_height / 2))

    # Start FPS timer
    timer = cv2.getTickCount()

    if not HEADLESS:...

    if STATE == "TRACK":
        if TO_TRACK is not None:...

        # Update tracker
        ok, bbox = tracker.update(frame)

        if ok:...
            # Tracking success
        else:
            # Tracking failure
            cv2.putText(frame, "Tracking failure detected", (100, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2)

    # Display tracker type on frame
    cv2.putText(frame, TRACKER + " Tracker", (100, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2)

    # Display tracker type on frame
    cv2.putText(frame, str(f_width) + "x" + str(f_height), (100, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2)

    # Calculate and Display FPS
    fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer)
    cv2.putText(frame, "FPS : " + str(int(fps)), (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2)
```

```python
if ok:
    # Tracking success
    p1 = (int(bbox[0]), int(bbox[1]))
    p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
    p_center = (int((p1[0] + p2[0]) / 2), int((p1[1] + p2[1]) / 2))

    # Draw bounding box
    cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)

    # Draw distance from center
    cv2.line(frame, f_center, p_center, (255, 0, 0), 2, 1)

    vector = (
        p_center[0] - f_center[0],
        p_center[1] - f_center[1]
    )
    normalized_vector = (
        vector[0] / (f_width / 2),
        vector[1] / (f_height / 2)
    )
    clamped_vector = (
        min(max(-1, normalized_vector[0]), 1),
        min(max(-1, normalized_vector[1]), 1),
    )
    scaled_vector = (
        int(clamped_vector[0] * 255),
        int(clamped_vector[1] * 255)
    )

    cv2.putText(frame, "Center offset: " + str(clamped_vector), (100, 110), cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                (255, 255, 0), 2)

    cv2.putText(frame, "to arduino: " + str(scaled_vector), (100, 140), cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                (255, 255, 0), 2)

    if arduino is not None:
        arduino.write(str(scaled_vector))
```

Performance of Object Tracking Algorithms

- different algorithms for object tracking available
  - KCF: fast, but cannot recover from occlusion of the object
  - CSRT: slower, but handles occlusion well

➡ tradeoff between accuracy and speed (even more on limited resources)

	KCF / 1920x1080	CSRT / 1920x1080	KCF / 1280x720	CSRT / 1280x720
Raspberry Pi	5 FPS	2 FPS	-	-
Intel i7	90 FPS	27 FPS	60 FPS	27 FPS
M1	140 FPS	40 FPS	180 FPS	72 FPS

Object Recognition - Current Problems

- selection of the Region of Interest needs a GUI
  - running an X session on Raspberry Pi would degrade performance even more
  - current solution: GUI on laptop with Websocket connection to Raspi
- where is the bottleneck for performance?
  - connection to camera? camera resolution?
  - object tracking algorithms?
  - do we get a better performance with OpenCV for C++ compared to Python?
  - replace raspberry pi
  - *overall:* how to improve our solution to achieve more accuracy as well as more speed

Object Recognition - Development Experiences

- remote development can be cumbersome
  - setup, debugging, SSH, sometimes also VNC
  - not enough RAM for X session together with remote development tool
  - “solution”: we do most development on our laptops (which also have webcams for testing)
- well documented (and actively developed) libraries are a blessing
  - what we often saw instead: last commit on 4th August 2013, 142 unresolved issues, exits with some unknown error message
- “hacking together” a solution often results in very convoluted code

  ➡ refactoring needed

3D-Printing parts

- to mount camera and motors, we need to 3D print some parts
- we are allowed to use the laser cutter and 3D printer of Prof Baudisch's lab
- however: many changes to original blueprint
  - ➜ Arduino Uno instead of Arduino Nano
  - ➜ 40 mm NEMA 17 instead of 22 mm NEMA 17
  - ➜ different camera
- ➜ need to change blueprint
- ➜ check usage of laser cutter for broader parts (mounts) and 3d printing for finer parts (gears, screws)

Next milestones

1. We want to 3D-print the parts that need to be 3D-printed and assemble the camera slider for at least one axis.
2. We want the motors to try keep the Region of Interest in the center of the frame.
3. We want to improve performance of our object recognition on Raspberry Pi.