Camera Tracking on Moving Objects using RPi + Arduino

Ella, Lukas, Philipp, and Daniel

Motivation

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

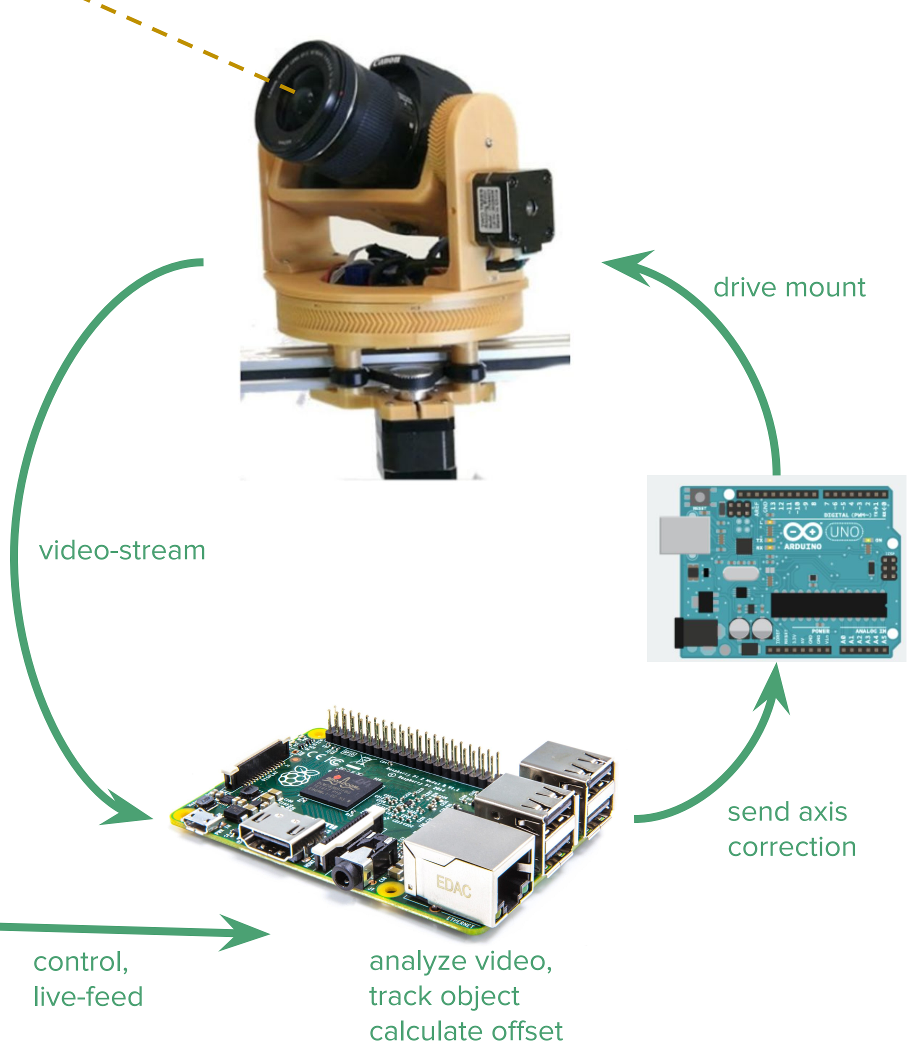
Drive mount using wireless joystick input

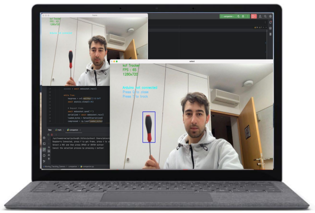
Project Goal

Spice things up slightly by:

- Using Raspberry Pi (OpenCV support) as controller
- Implementing a simple tracking algorithm
- Limit to two axis

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





Implementation

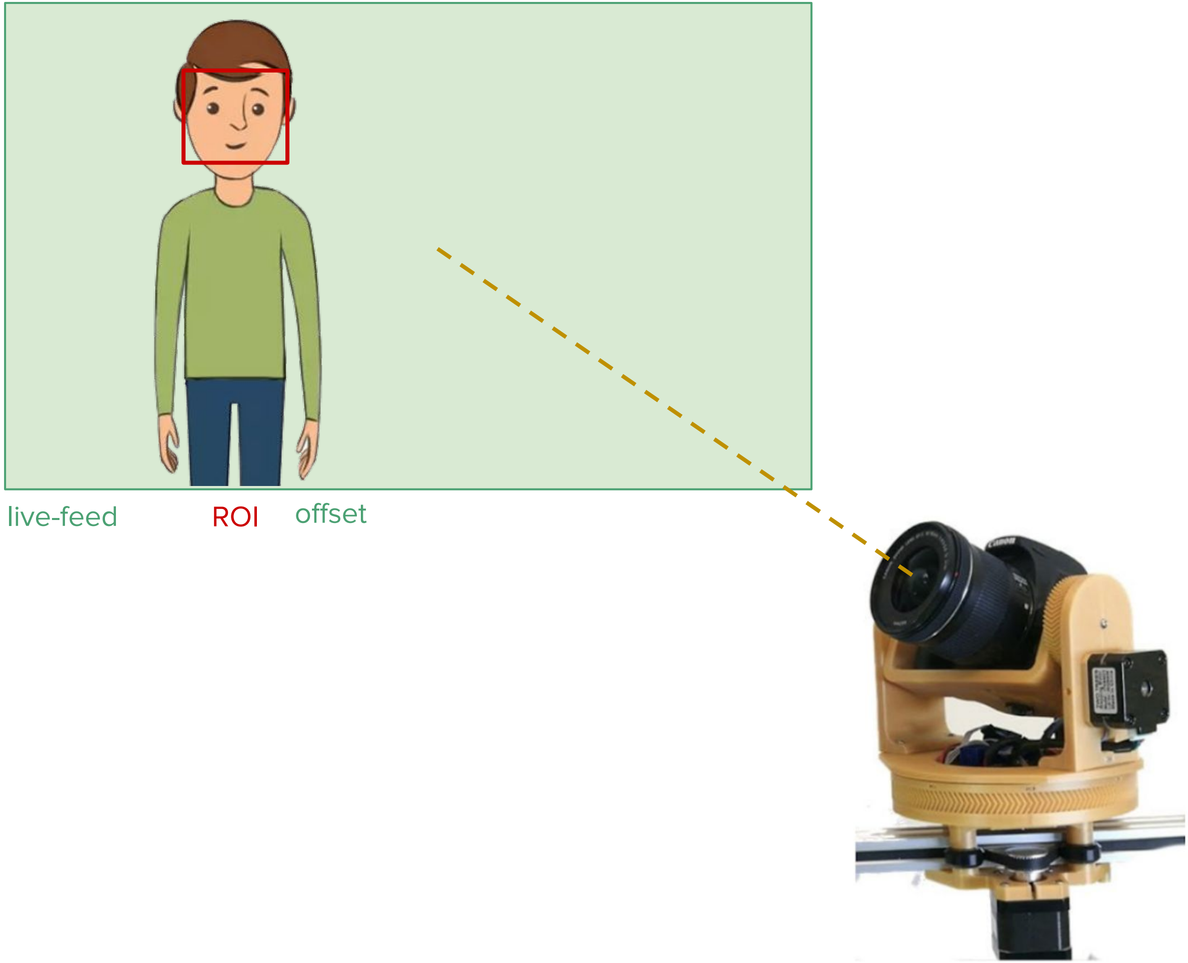
Three subtasks:

1. Object Tracking (Raspberry Pi)
2. Motor Control (Arduino)
3. Build Camera Bot

Object Tracking

User Journey

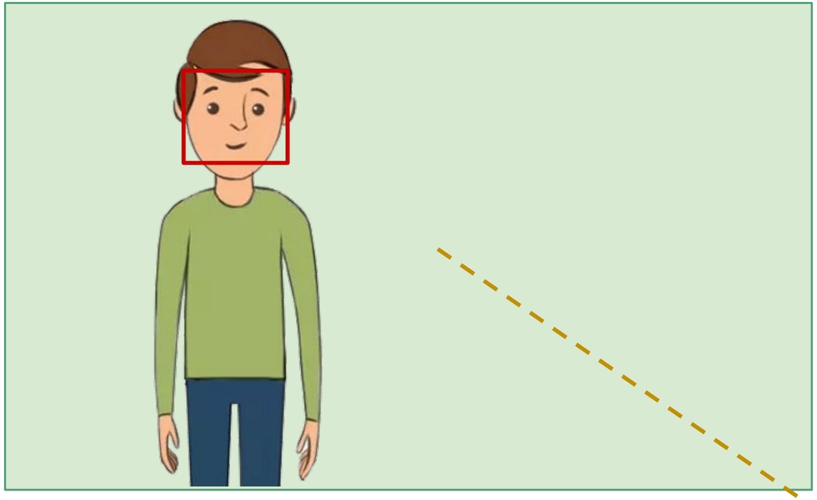
1. Start Raspberry
   [RPi opens a websocket server]
2. Start Companion
   [Companion connects to server to get a live-feed]
3. User selects ROI to track
4. Camera “follows”



Object Tracking

User Journey

1. Start Raspberry
   [RPi opens a websocket server]
2. Start Companion
   [Companion connects to server to get a live-feed]
3. User selects ROI to track
4. Camera “follows”



live-feed ROI offset

f: get frame

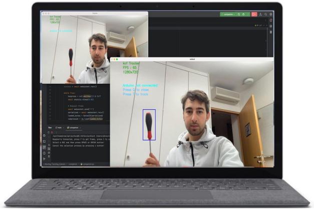
c: connect arduino

t: select ROI request

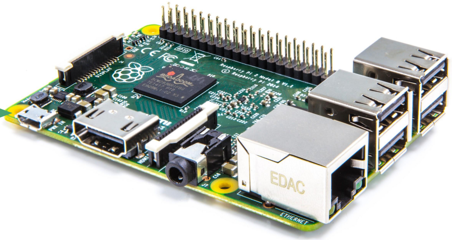
q: exit

control bytes “f”, “c”, “t”, “q”

compress to jpg and send as stream



companion

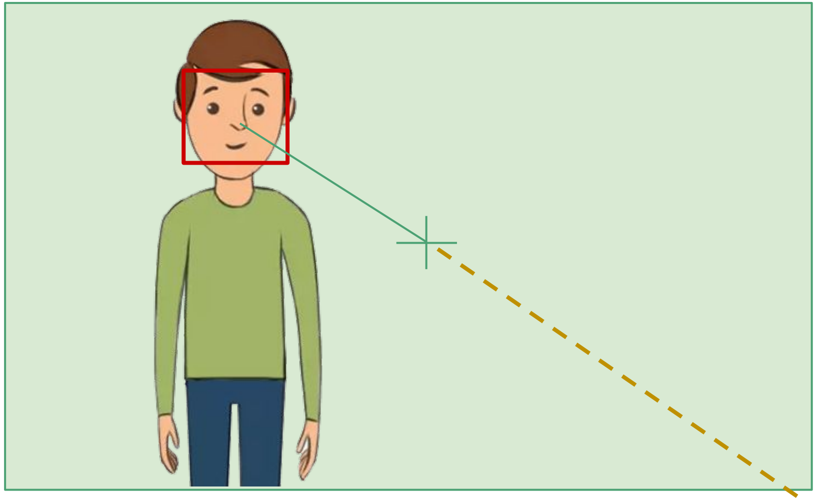


analyze video,
track object
calculate offset

Object Tracking

Objective:
Keep ROI in the center of the frame

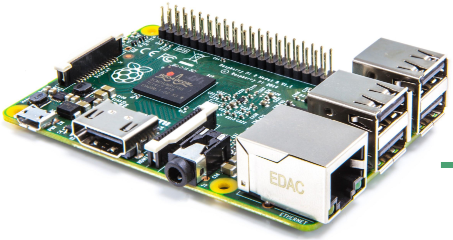
1. Track ROI
2. Calculate offset from center
3. Normalize according to frame size
4. Discretize into 10 velocity buckets per axis
5. send correction vector to Arduino



live-feed ROI offset

Ingredients RPi

- OpenCV
- CSRT-Tracker
- WebSocket on separate thread
- Serial communication on separate thread

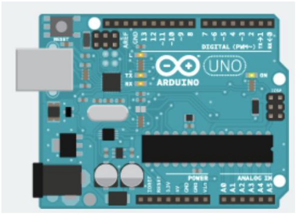


track object

[x_velocity, y_velocity]

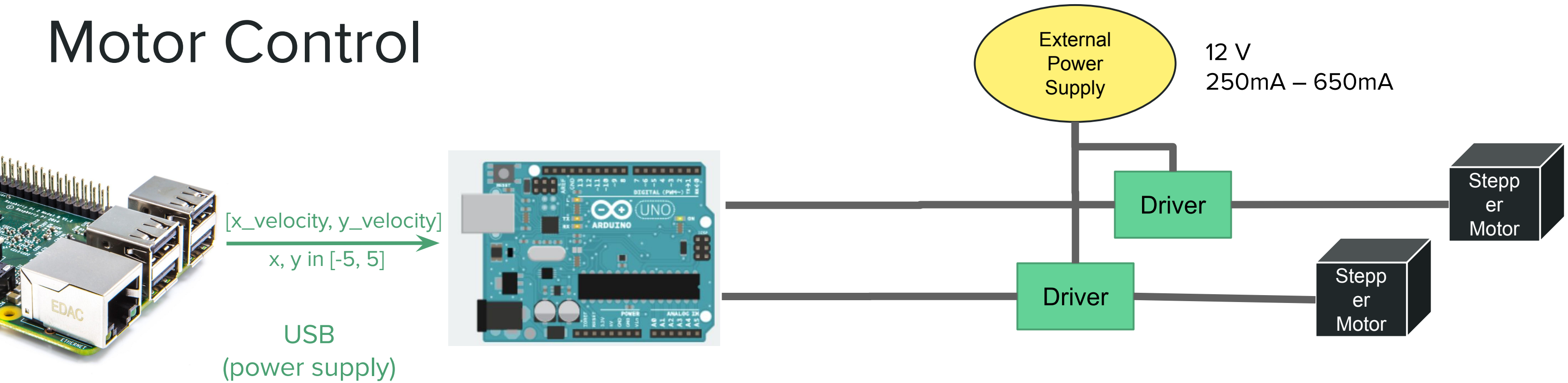
x, y in [-5, 5]

USB

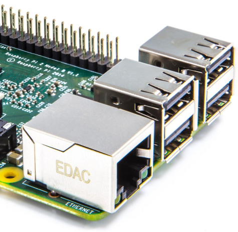


send axis correction

Motor Control



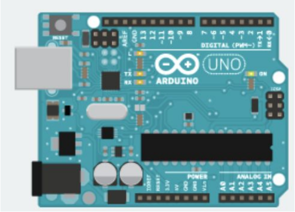
Motor Control

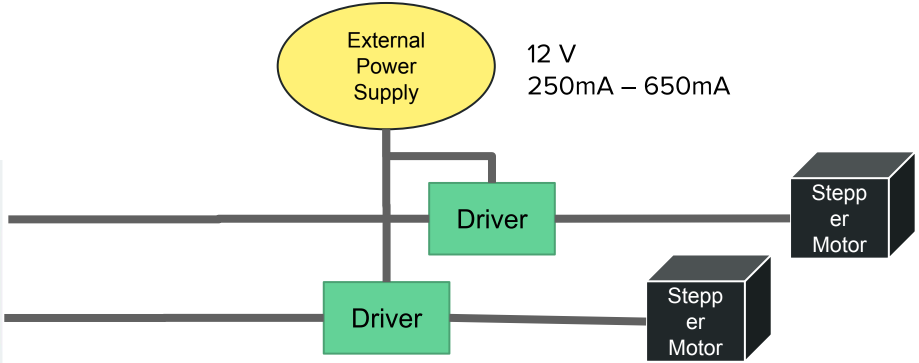


[x_velocity, y_velocity]

x, y in [-5, 5]

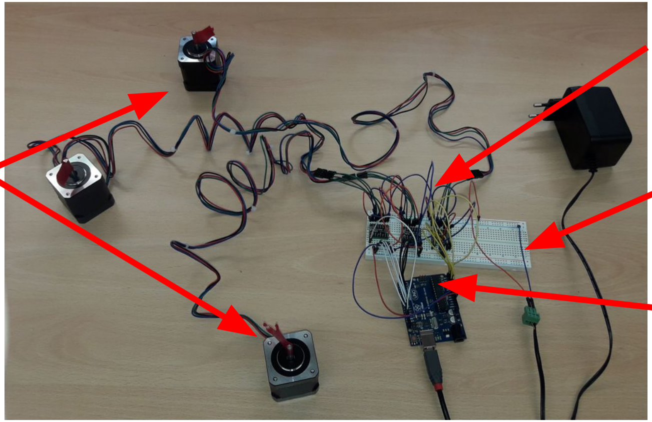
USB
(power supply)





12 V
250mA – 650mA

Nema 17 Stepper Motors



Stepper Drivers (TMC2209)

External Power Supply (12V/1

Arduino Uno

Since Mid-Presentation

C++ Rewrite

Motivation:

- Only 1-2 FPS on RPi using Python and OpenCV

Attempt:

- Rewrite RPi application in C++

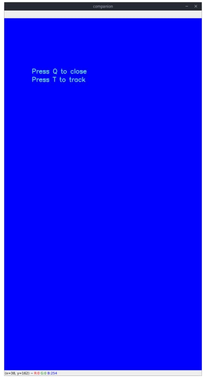
C++ Rewrite

Motivation:

- Only 1-2 FPS on RPi using Python and OpenCV

Attempt:

- Rewrite RPi application in C++



Journey:

- hard to interop
  OpenCV for C++ with OpenCV for Python companion
- different multithreading paradigms
  concurrent programming in C++ is harder than it seems at first glance
- pointer arithmetic and seg faults

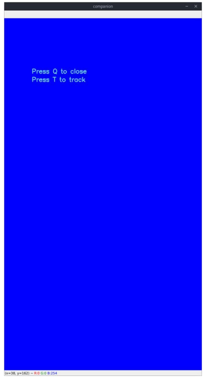
C++ Rewrite

Motivation:

- Only 1-2 FPS on RPi using Python and OpenCV

Attempt:

- Rewrite RPi application in C++



Journey:

- hard to interop
  OpenCV for C++ with OpenCV for Python companion
- different multithreading paradigms
  concurrent programming in C++ is harder than it seems at first glance
- pointer arithmetic and seg faults

Result:

- ~4 FPS

Lessons

1. When debugging, check **all your assumptions**! They’re probably wrong.
2. Put I/O intensive and compute intensive tasks into **separate threads**.
3. If your devices need WiFi for communicating with each other, set up your own **lab WiFi** instead of using smartphone hotspots. It will save you hours of debugging and makes port forwarding much easier.

C++ Rewrite

Main Loop

```
while(STATE != TrackingState::TERMINATE) {
    capture >> frame;
    if(frame.empty()) break;
    cv::Point2d f_center(frame.cols/2.0, frame.rows/2.0);
    int64 timer = cv::getTickCount();

    if(STATE == TrackingState::TRACK) {
        if(TO_TRACK) {
            auto [reference_frame, bbox] = *TO_TRACK;
            tracker->init(reference_frame, bbox);
            TO_TRACK = std::nullopt;
        }
        if(tracker->update(frame, bbox)) {
            cv::Point vector = calculateOffsetVector(bbox, f_center);
            displayTrackingSuccess(frame, bbox, f_center, vector);
            std::lock_guard<std::mutex> lock(VECTOR_MUTEX);
            OFFSET_VECTOR = vector;
            DO_SEND_VECTOR = true;
        } else {
            DO_SEND_VECTOR = false;
            displayTrackingFailure(frame);
        }
    }

    int fps = cv::getTickFrequency() / (cv::getTickCount() - timer);
    displayGeneralInfo(frame, fps);
    ACTIVE_FRAME = frame.clone();
}
```

Websocket

```
void websocket_handler(server* serv, websocketpp::connection_hdl hdl, message_ptr msg) {
    std::string message = msg->get_payload();

    if(STATE == TrackingState::RECEIVE_BBOX) {
        std::cerr << "Track region received\n";
        std::vector<int> bbox(4);
        message = message.substr(1, message.length()-2);
        for(int i = 0; i < 4; i++){
            size_t pos = message.find(',');
            bbox[i] = std::stoi(message.substr(0, pos));
            message.erase(0, pos+1);
        }
        cv::Rect roi(bbox[0], bbox[1], bbox[2], bbox[3]);
        TO_TRACK.emplace(LAST_SEND_FRAME.clone(), roi);
        STATE = TrackingState::TRACK;
    } else if(message == "c") {
        std::cerr << "Companion is connected\n";
        serv->send(hdl, "ok", opcode::TEXT);
    } else if(message == "f") {
        std::cerr << "Frame requested\n";
        LAST_SEND_FRAME = ACTIVE_FRAME.clone();
        std::vector<uchar> buf;
        cv::imencode(".jpg", LAST_SEND_FRAME, buf);
        serv->send(hdl, buf.data(), buf.size(), opcode::BINARY);
    } else if(message == "t") {
        STATE = TrackingState::RECEIVE_BBOX;
    } else if(message == "a") {
        connectToArduino();
    }
}
```

Serial

```
void sendToArduinoThread() {
    while(STATE != TrackingState::TERMINATE) {
        if(ARDUINO_STATE != ArduinoState::CONNECTED) {
            std::this_thread::sleep_for(std::chrono::milliseconds(100));
            continue;
        } else if(DO_SEND_VECTOR) {
            cv::Point vector_to_send(5, 5);
            {
                std::lock_guard<std::mutex> lock(VECTOR_MUTEX);
                vector_to_send = OFFSET_VECTOR + cv::Point(5, 5);
                DO_SEND_VECTOR = false;
            }
            std::string message = std::to_string(vector_to_send.x) + std::to_string(vector_to_send.y);
            ARDUINO->write(message);
            std::cerr << ARDUINO->readline();
        }
    }
}
```

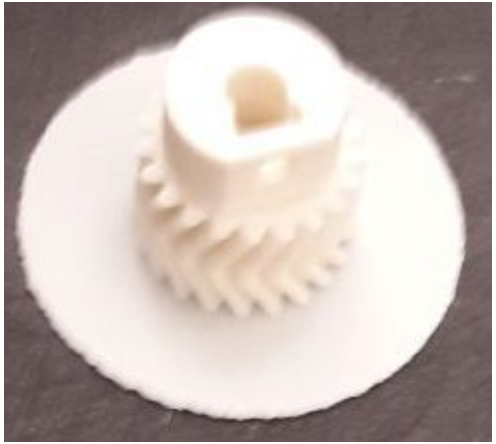
Arduino-to-Raspberry communication

- Serial library in Python/C++ to connect Raspberry to Arduino
- Arduino has built-in support for Serial
- communication protocol:
    - current position of the object on the frame communicated as integer coordinate between (0,0) and (10, 10)
    - (5,5) is the middle of the frame
    - **advantage:** only need to transmit two digits, but accurate enough for our use case

```
void loop() {

    for (int i = 0; i < sizeMotors; i++) {
        steppers[i].run();
    }

    if (Serial.available() > 1) {

        // 0-9 allowed
        char x = Serial.read();
        char y = Serial.read();

        // convert chars to int. Substract 5 to put 0 in the middle
        int xVelocity = (x - '0') - 5;
        int yVelocity = (y - '0') - 5;

        steppers[0].setMaxSpeed(10 * abs(yVelocity) * steps_per_mm);
        steppers[1].setMaxSpeed(10 * abs(xVelocity) * steps_per_mm);

        int sign0 = yVelocity > 0 ? 1 : -1;
        steppers[0].move(1 * steps_per_mm * sign0);
        int sign1 = xVelocity > 0 ? 1 : -1;
        steppers[1].move(10 * steps_per_mm * sign1);
    }
}
```

Building the CameraBot

- Initially used the 3D printer in Prof Baudisch's lab
    - ➜ Ended up taking too much time, so we outsourced
- Had to make changes to original blueprint
    - ➜ Arduino Uno instead of Arduino Nano
    - ➜ 40 mm NEMA 17 instead of 22 mm NEMA 17
    - ➜ Different camera size
    - ➜ Different scope
        - no slider axis but feedback loop



This part of took more than 1h to print
(in good quality)

The long tale of fast custom printing

5.01.2023: Order of 3D parts (estimated delivery time 5-7 days)

18.01.2023: Asking again per mail what the status is

18.01.2023: ~"Will send tomorrow. We had difficulties printing one part"

24.01.2023: Automatic email confirming delivery

26.01.2023: Arrival of package

The package.

3D-printing: Assembly

Learnings:

- Parts are not 100% accurate
- Parts are not completely round
- Tight assembly increases friction

- Moving parts have to be thinned by file and smoothed by sand-paper or aceton
- 3D printed bearings have a lot of friction

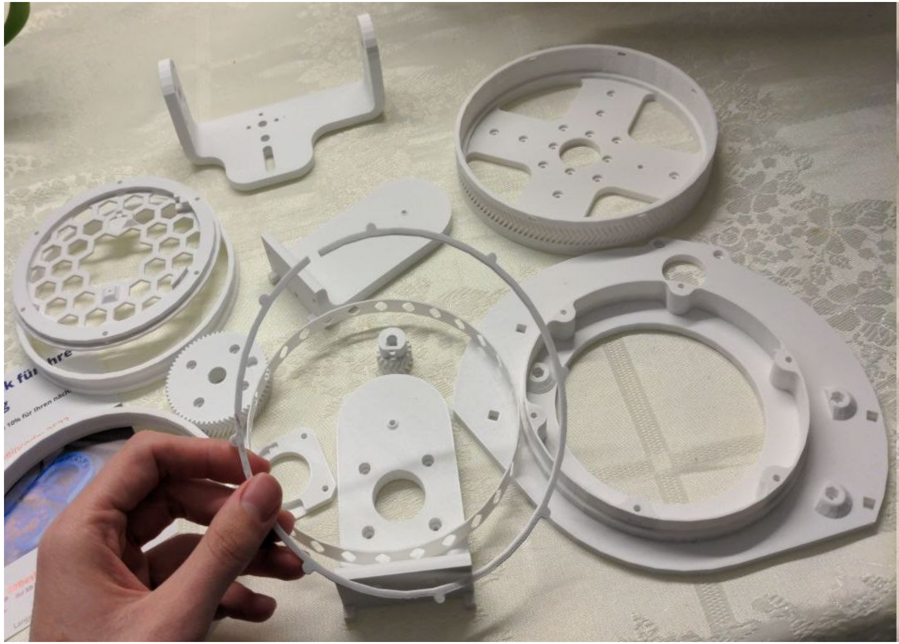
Putting it all together

- Motors didn’t move setup including camera
  - Measured amperage using measurement unit within motors circuit in iot lab
  - Initially used 200 mA
- Changed RMS-Current in code up to 500 mA
- Interesting: Stepper motors use base ampere rate at rest

video: see
https://drive.google.com/file/d/17-qmVV_L9QbTcYhCXW39X6Ke_6763iK9/view?usp=sharing

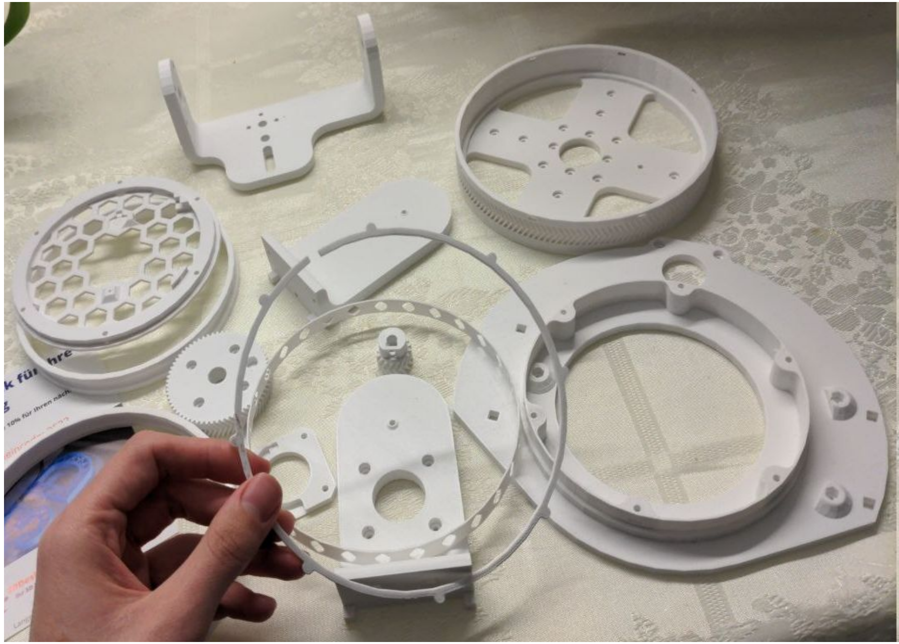
Hardware Projects - Learnings

- Hardware iteration cycles take long
- Surprise: Software iteration cycles take long
  - Development experience on Arduino and RaspberryPi is poor
- Solution:
  write for PC, test and debug on PC,
  then adapt for RPi
- We didn’t emulate an arduino:
  Painful debugging experience
- Measure everything to be certain



Hardware Projects - Learnings

- hobbyist and hardware stuff can be really badly documented
- the hardware ultimately sets the limit of a device’s performance
- 3D printing takes a long time, but ordering 3D printed
  parts might take even longer



Evaluation

Object Recognition: Performance of Algorithms

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

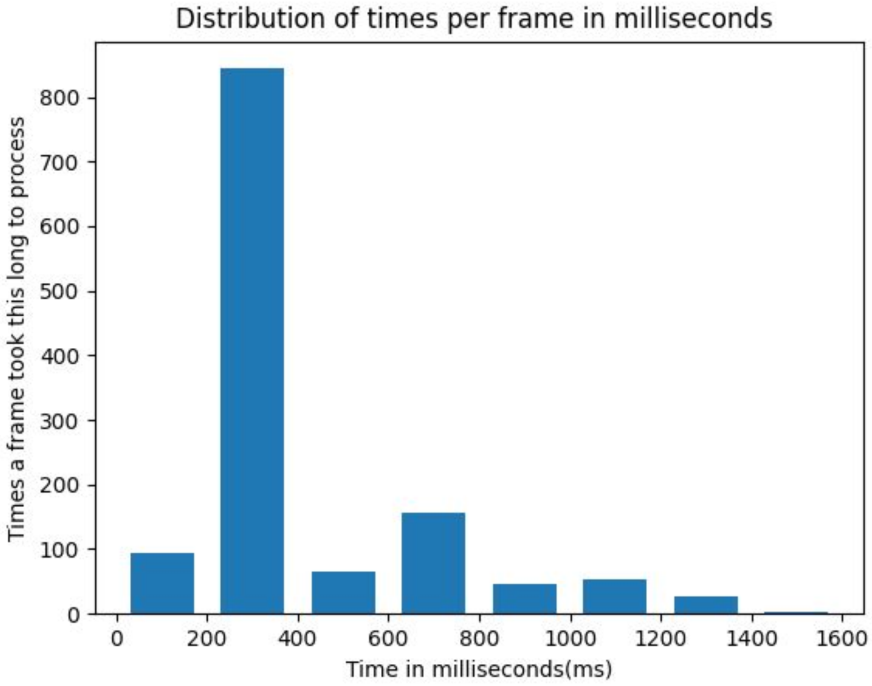
➔ tradeoff between accuracy and speed (even more on limited resources)
➔ we ended up electing CSRT

Learning:

- CSRT is ok for occlusion but often can’t recover if ROI was lost

Object Recognition: Performance

- we switched from Python to C++ because of better performance ➜ in our tests, performance is still around a third of a second in most cases
- C++ on RPi
- Get around 3-4 FPS



Learning:

- C++ is faster, we assume because it's compiled and doesn't use garbage collection
- The difference to Python is not huge: Bottleneck was hardware (no GPU for CV)

Last time’s milestones: How far did we get?

- ✅ 3D-print and assemble the camera slider for at least one axis
- ✅ Motors should keep the Region of Interest in the center of the frame
- ✅ Have a realistic user journey
- ✅ Evaluate performance of implementation and find potentials for improvement

What future teams could add to the project

- ❎ further improve usability
- ❎ further improve performance software-wise
- ❎ more reliable ROI tracking
- ❎ use better hardware (like Nvidia Jetson)
- ❎ use more fault-tolerant communication protocols
- ❎ more fine-grained motor control
- ❎ support additional axes

Video