WPI

Last modification: November 13, 2024

RBE 2002 (B-24)
Lab 4
Computer Vision

Be sure to follow along with the Worksheet at the end of this document as you do the activities.

You will work in pairs to complete the activities and Worksheet. Do not take a 'divide-and-conquer' approach – it is important that each student masters the material.

1 Introduction

For your final project, your robots will need to use visual cues both to navigate (aligning with a garbage can) and to gather information (using an ID to "bill" the correct customer).

In this week's activities, you'll add the lifter mechanism to your robot, which has an integrated camera mount for the OpenMV camera provided.

Be gentle with the lifters. The load cells are rated for 1gk max, which means that if you subject them to significantly more stress (typically twice the rating), you will damage the internal strain gauges. **Remove the load cell while you are experimenting with the camera.** You can reattach it next week, when you lift and weigh a garbage can.

There is a base set of activities to earn 90 points for the lab. To receive a higher score, you will need to go beyond expectations and complete one or more extra activities. Suggested extra activities can be found near the end of this document.

2 Background

2.1 OpenMV

The OpenMV camera is a low-cost, extensible, machine vision module designed to "bring machine vision algorithms closer to makers and hobbyists" – OpenMV strives to be the "Arduino of computer vision." Though we will often refer to it simply as a "camera," the OpenMV breakout includes not only the camera, but also a powerful on-board processor. In addition, the OpenMV IDE includes a number of example codes to do everything from face detection to optical flow to much more. Though we won't be using the functionality here, the creators have even tied it in with the Edge Impulse machine learning engine to do powerful image recognition. You are encouraged to load some of the sample programs onto the device and try them out!

2.2 AprilTag

Or Apriltag. Or April Tag. You'll see it presented many different ways.

AprilTag is "a visual fiducial system, useful for a wide variety of tasks including augmented reality, robotics, and camera calibration." One can easily print a variety of AprilTags and use them as landmarks for navigation or for detecting other robots. In your final project, your Romi will use tags to align itself with garbage cans and, through reading the IDs, to bill the correct customer.

2.3 Communication with the Romi

The OpenMV camera is programmed using MicroPython, a variation of Python 3 that is optimized for running on microcontrollers. We have modified the OpenMV example scripts to allow the camera to communicate with your robot – there is a version that communicates via the UART and another that uses I2C. (Alas, the pins on the SPI bus on the Romi are dedicated to other purposes, so we can't build a connection using SPI). The UART version is the easiest to manage, though you are encouraged to explore the I2C version in the extra activities. There are instructions for wiring and implementation in the libraries.

Otherwise, we have covered the necessary background in class – see the slides from the relevant lectures – but be sure to ask questions if you have them!

Lab Activities

3 Exploring the OpenMV Camera

As noted, the OpenMV camera can be used to detect April tags, among other things.

Procedure

1. Following the instructions on the OpenMV page, install the OpenMV IDE.
2. Upload the `find_apriltags.py` example found **in the OpenMV Examples that are built into the OpenMV IDE**. When you connect your camera, you will likely get a notification that the firmware can be updated. You can safely ignore the notification; alternatively, ask your instructor to help you with the update – if it goes wrong, the camera can easily be bricked. We can likely unbrick it, but it takes a few minutes to get it working again.
3. Navigate to this page, which has images of several tags. Or find a physical print out of an April tag. **Note**: the most recent example code will only detect tags from the `36H11` family of tags.
4. Point the camera at the tag. In the OpenMV IDE, check the camera view to see if it can identify a tag. If you open the Serial Terminal in the OpenMV IDE, you should be able to get basic data about each tag that it detects.
5. Do an experiment to determine the orientation and extents of the camera. Draw the pixel coordinates on the figure in the Worksheet, including the pixel coordinates of the center and corners of the field of view.
6. Explore at least one other example program in the OpenMV IDE.

4 Connecting the OpenMV Camera to the Romi

Here, you will connect a camera to a Romi and report back data about tags through the microprocessor on the Romi.

4.1 The OpenMV Camera with UART

The UART connection is the easiest to manage for the camera. In short, the chip on the Romi simply listens for AprilTag information from the camera. The connection is one-way, as there in nothing that needs to be sent from the Romi to the camera. The one drawback is that the WiFi code also uses the UART, but so long as you don't want to *send* info to the Romi, you'll be fine.

Procedure

1. Add the `openmv-apriltags` repository to your `platformio.ini` file under the `lib_deps` section:

```
lib_deps =
...
    https://github.com/WPIRoboticsEngineering/openmv-apriltags
...
```

2. Grab the `apriltags_uart.py` program from the repository and upload it to your camera. The program is similar to the example from Section 3, but has a UART interface built into it. Verify that it is detecting tags in the OpenMV IDE.
3. Connect the pin `TX` on the OpenMV camera to the pin `RX1` on your Romi. Check the Python script for the camera to determine which UART you'll use on the OpenMV. The pinout will come in handy. Note that the OpenMV is 5V tolerant, so there is no need for level shifting. Don't forget to share a ground. Do not connect power from the Romi at this point – you will use two USB cables, one to the Romi and one to the camera, so that you can verify performance.
4. Upload the `apriltag_finder_uart` example to your Romi. This program monitors `Serial1` to see if new data is available and prints to the Serial Monitor if there is. Check that `Serial1` is set at the correct baud rate (check the Python script for the baud rate).
5. Point the camera at some AprilTags. The program running on the Romi should print out successful tag finds to the Serial Monitor in `VSCode`.

4.2 State Machine

Here, we concentrate on the parts of the state machine that are relevant to AprilTag detection. In the final project, your robot will need to drive to a location and then search for tags, but for this lab, you'll simply command the robot to start searching from the `IDLE` state.

The basic plan is to have your Romi do the following:

- Upon command, start searching for tags by slowly rotating in place.
- Upon detection of a tag, start approaching the tag by using a P(ID) controller to align with the tag.
- Upon getting "close enough," stop the Romi. We'll deal with actual lifting next week.

You might want to read ahead in Section 4.3 to help define your state machine.

Procedure

1. Draw out the state transition diagram described above *before* you get started coding.

4.3 Aligning with a tag

Here, you will add code to allow your Romi to align itself with a tag. You will need to create at least a proportional controller to control both distance and heading of the Romi.

Procedure

1. Add two states to your `Robot` class:
   - `ROBOT_SEARCHING`, which will be used to start looking for a tag, and
   - `ROBOT_APPROACHING`, which will be used to align with the tag, once one has been found.
2. Create a file, `robot-cam.cpp`, to manage the camera aspects of your `Robot` class. Add the following methods to the `Robot` class:
   - `EnterSearchingState()`, for starting to search,
   - `EnterApproachingState()`, for starting to approach,
   - `HandleAprilTag()`, used to process a tag.
   - `CheckApproachComplete()`, which is used to determine if the robot has reached the point where the garbage can be lifted.

   Skeleton templates for these functions are given in the Appendix.
3. Add code to use a button or the IR remote to put your Romi in the searching state.
4. Add a checker/handler for AprilTags to the end of `RobotLoop()`, just like the other checker/handler functions. As discussed in class, `OpenMV::checkUART()` is passed a tag *by reference*, so that it can return complex information. You will need to check the return value of that function to know if a tag has been detected.
5. Still with two USB cables, *carefully* test your code.

5 Powering the Camera from the Romi

To use the OpenMV with your Romi without a second USB cable, you'll need to follow the instructions in save your script to your OpenMV Cam, which will allow the script to run without being connected to a computer. After loading the script, *wait for the LED on the OpenMV to turn off or you will brick it*. Then, **disconnect all power from both devices** and add a connection from 5V on your Romi to the `VIN` pin on the camera. **Have someone independently check your wiring.** The cameras are expensive! Reconnect the USB to the Romi (but not the OpenMV board). Verify that you can still detect the tag.

6 Verification

Perform tests to determine/ensure the reliability of your Romi. Perform at least 10 iterations to test the following performance metrics,

- Successful detection of the tag and identification (correct ID),
- Length of time between initial detection and final alignment, and
- The error in alignment. If you do not have a complete lifter (which is best), then add a piece of cardboard or popsicle stick (if you happen to have one) to simulate the length of the lifter. Measure the error from an "ideal" point relative to the AprilTag.

Compile your data in the table in the Worksheet.

Extra Activities

7 Lost Tags

What happens when you pull the tag away from the camera? If you haven't accounted for the loss of a tag, then your Romi probably just keeps doing whatever the last command was. But because the camera takes time to process snapshots, your checker returns `false` even when there is a tag in the field of view, so you can't simply tell it to stop when no tag is detected.

Better is to use a timeout to indicate when you've lost a tag. For example, you can set a timer to 1 second and reset it every time your Romi sees a tag. If the timer expires, then the Romi hasn't seen a tag for at least 1 second (a long time in Romi world), you can handle the loss of a tag.

Note: If you use your professor's `EventTimer` class, you should call `Start()`, not `Restart()`, since the latter will add time to the end of the current period – i.e., calls to `Restart()` compound.

8 The OpenMV Camera with I2C

The UART works well for communicating with the camera. It is a lightweight protocol and since the communication is only one direction, the Romi only needs to "sit and listen" for incoming tag information.

The downside is that the UART is taken up by the camera, so it's much more difficult to connect other devices, say the `ESP32` if you wanted to receive information wirelessly (if you only want to send, you're safe). Since the I2C bus is already set up for the IMU, it's easy enough to use it to connect to the camera.

Here, you can set up the camera through I2C. Like the IMU, communication is done through polling, but it is straightforward to add an interrupt line to let the Romi know that tag data is available.

Procedure

1. Grab the `apriltags_i2c.py` program from the repository and upload it to your camera. The program is similar to the example in the pre-lab, but has an I2C interface built into it. The code will run a little slow (just a couple frames per second) when it's not talking to another chip, but you can still verify that it is detecting tags.

2. Connect the OpenMV camera to the Romi using I2C. Don't forget to share a ground, but *not* power at this time. The pinout will come in handy. Check the sample python script to see which `I2C` bus to use on the camera.

3. Upload the `apriltag_finder_i2c` example to your Romi. This program repeatedly polls the camera to see if new data is available and prints to the Serial Monitor if there is.

   Warning! The code implies that it can manage multiple tags, but it will not work as is. You can probably get it to see multiple tags by changing `if(tagCount)` on line 20 to `while(tagCount--)`, but that's untested.

4. Point the camera at some AprilTags. The program running on the Romi should print out successful tag finds to the Serial Monitor.

8.1 Camera Notifications

The I2C example code uses polling to determine if there are tags visible. As with the IMU, polling is wasteful since most transactions are fruitless. While the IMU does have an interrupt pin, it's not broken out on the Romi (a seriously missed opportunity by Pololu). The camera, on the other hand, has several exposed pins, most of which could be used to tell the Romi when it has new data. You could, for example, edit the OpenMV script to set a pin HIGH when it has a tag. Then on the Romi, you could respond to the notification in an interrupt by raising a flag on the Romi (do **not** try to read the tag from inside an ISR – just raise a flag and handle it in the main loop). Polling would also work fine here, since you only need to check the status of a pin). Only when the flag is raised do you need to read the tag, which conserves resources. Be sure to lower the flag on the OpenMV when the tag is read.

Describe your implementation (with references to your code) and the advantages/disadvantage of your method.

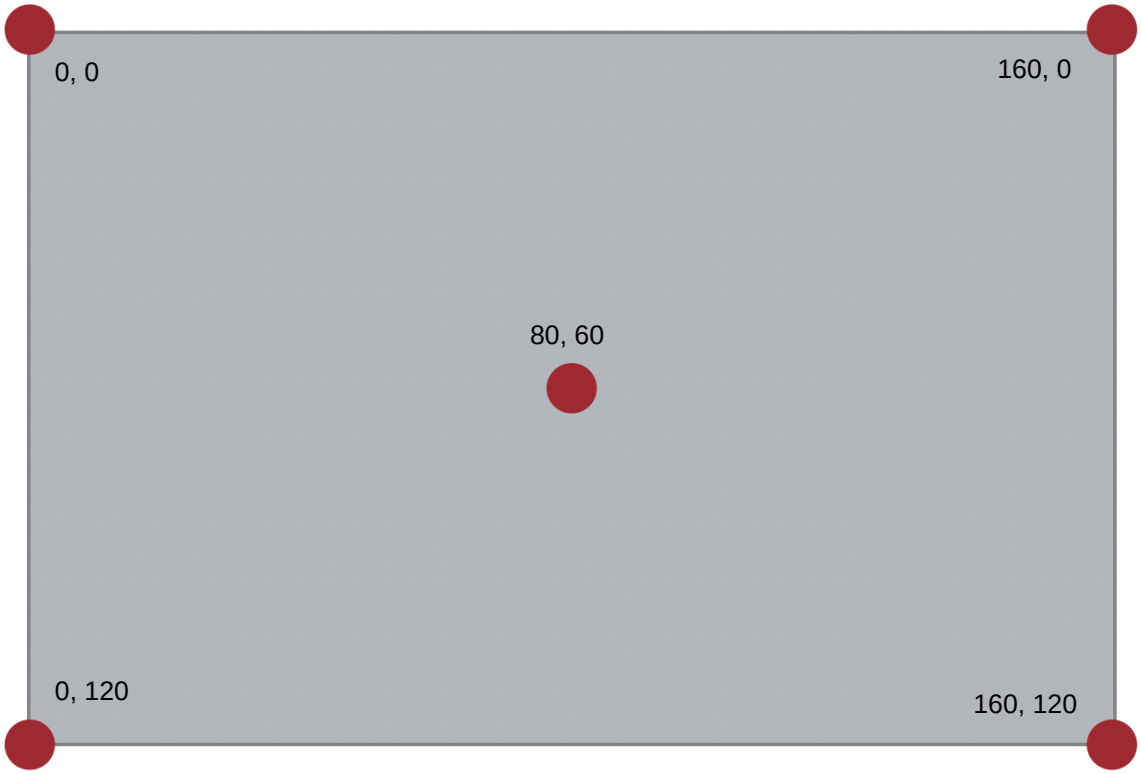
9 Indicate the ID of the tag

While you can leave the USB cable attached for the activities, having a way to indicate the tag ID would be useful. You could, for example, blink an LED to indicate the tag ID, but note that you must do that with non-blocking code. A software timer would work well. If you want to dig into hardware timers, that can also work. You could also put a counter where you check the Chassis timer – it fires every 20 ms, but you can skip some to make it countable by humans. If you want to use WiFi, see your instructor.

Worksheet

Question

Annotate the figure below to describe the field of view of the OpenMV camera. Label the axes as if seen from ”inside” the sensor. Label the coordinates of the corners and center pixel.



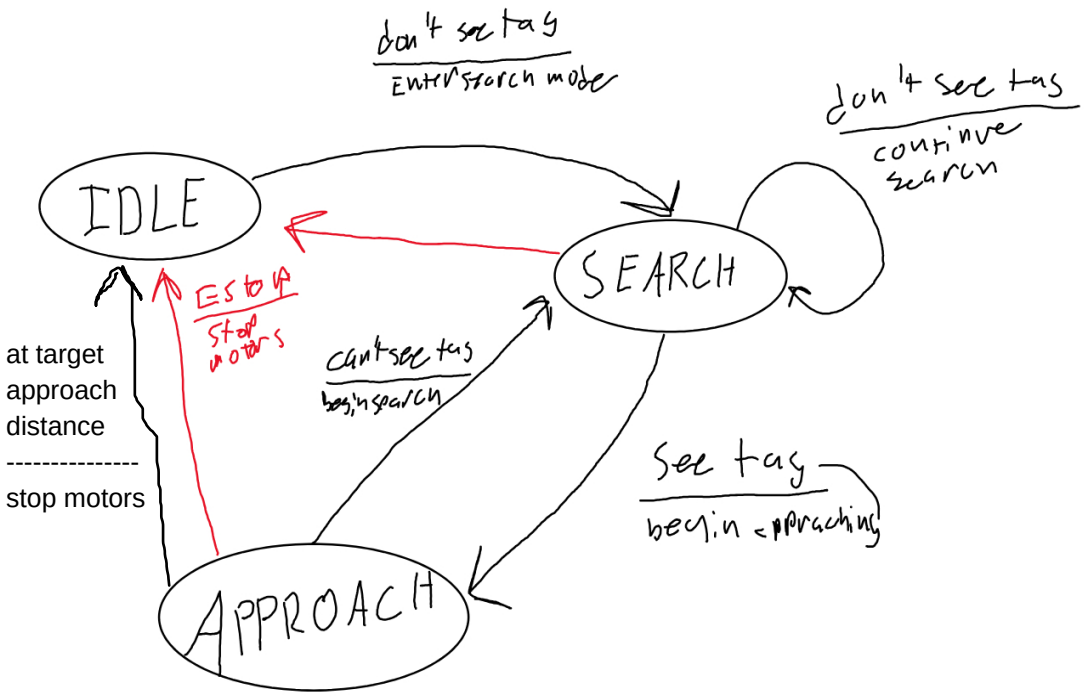
Question

What other OpenMV example did you explore? How could the methods/algorithms be useful for a robot?

We tested the example that found the relative pose of the robot compared to the apriltag.
This could prove to be useful in cases where you want to follow a more complex path to approach the tag, instead of just following a straight line.

Question

Draw out state transition diagrams for aligning with the AprilTag.



Overall, the error seems to be reasonable, the error values we collected were the error in the height of the tag in the frame and the pixels of the tag in the horizontal field of view of the camera. This gives us an accurate picture of what the tolerance of the alignment is.

Question

Run	ID Correct?	Time to Alignment	Alignment Error	Notes
1	yes	~ 3 seconds	dist: 1, rot: 0	
2	yes	~ 2 seconds	dist: 0, rot: 0	
3	yes	~ 3 seconds	dist: 0, rot: 1	
4	yes	~ 1 seconds	dist: 0, rot: 0	
5	yes	~ 2 seconds	dist: 2, rot: 1	
6	yes	~ 3 seconds	dist: 1, rot: 0	
7	yes	~ 1 seconds	dist: 0, rot: 1	
8	yes	~ 2seconds	dist: 1, rot: 1	
9	yes	~ 3 seconds	dist: 1, rot: 2	
10	yes	~ 2 seconds	dist: 2, rot: 0	

Question

Describe how your robot approaches the AprilTag. Reference the relevant functions/lines in your code release.

If the openmv uart detector detects a tag, it calls HandleAprilTag() and sets the state to ROBOT_APPROACH. The robot runs a PID controller on the height of the tag in the frame, and the distanceof the tag from the center of the frame. (robot.cpp function checkApproachComplete())

Question

Create a release of your repository as `lastname-firstname-week04`.

Question

Create a video of your robot completing the challenge and submit a clickable link here.

10 Wrapping up

Question

Team Member Contributions

1. Independently, describe how you contributed to the design tasks for this week's activities.

Student A Vishesh Kunduru

Student B Elliot Scher

2. For the core activities, assign a weight (as a percentage) to each team member's contribution. The core activities will be worth a total of 90 points. To receive higher marks, you will need to complete one or more extra activities.

Student	Percentage
Vishesh Kunduru	50%
Elliot Scher	50%

3. If either of you went beyond expectations for this lab, describe any extra work that you did. There is no requirement that both students work on the same activities, so be sure to identify who did what. Include evidence, either as a video or with code snippets or both.

Both of us did testing with the detection in greyscale and full RGB color. We found that the framerate decreased significantly (from 35 to 25 FPS) when running the camera pipeline in color. Interestingly, the detection was significantly improved when in color. There was noticeable increase in tag recognition consistency at farther distances when in color as opposed to grayscale. We found that this tradeoff doesn't make it worth it to run the camera with RGB, as we don't need the extra range that it gives us.

We both also implemented the "Lost Tag" functionality from the extra activities.

Appendix

11 Skeleton Code

```
void Robot::HandleAprilTag(const AprilTagDatum& tag)
{
    //You may want to comment some of these out when you're done testing.
    Serial.print("Tag: ");
    Serial.print(tag.id); Serial.print('\t');
    Serial.print(tag.cx); Serial.print('\t');
    Serial.print(tag.cy); Serial.print('\t');
    Serial.print(tag.h);  Serial.print('\t');
    Serial.print(tag.w);  Serial.print('\t');
    Serial.print(tag.rot); //rotated angle; try turning the tag
    Serial.print('\n');

    /** TODO: Add code to handle a tag in APPROACHING and SEARCHING states. */
}

void Robot::EnterSearchingState(void)
{
    /** TODO: Set Romi to slowly spin to look for tags. */
}

void Robot::EnterApproachingState(void)
{
    /**
     * TODO: Turn on the LED when the Romi finds a tag. For extra points,
     * blink out a number that corresponds to the tag ID (must be non-blocking!).
     * Be sure to add code (elsewhere) to turn the LED off when the Romi is
     * done aligning.
     */
}

/** Note that the tolerances are in integers, since the camera works
 *  in integer pixels. If you have another method for calculations,
 *  you may need floats.
 */
bool Robot::CheckApproachComplete(int headingTolerance, int distanceTolerance)
{
    /** TODO: Add code to determine if the robot is at the correct location. */
}
```