Dear Future Student,

If you are working with the Armcam code I wrote, this is a little bit of background on how the entire thing works, so hopefully you'll be able to add your own work.

I am assuming that you know the basics of ROS programming. In case you don't, all you need to know is that publishers send out data and that data can be caught by any subscriber tuned into the same message. So, in this program, most messages originate in the GUI file. That file will publish commands to the arm, which are caught in the control_arm.py file. Any movement commands are caught here. The only messages that aren't caught in that file, are the coordinates from the camera. Those are all published by the arm_cam.py file and caught in the box_pickup.py file which then publishes the same kind of movement commands as the GUI.

Now for the current state of the project:

The arm controls all work. You can send coordinates, open and close the gripper, and automatically pick up the bright yellow PLA cubes from the platform. The black tape shows where the cubes can go, and they need to be visible before you try doing the Auto-Pickup button. The code does well detecting the yellow cubes and the blue tape on the arm. The yellow allows it to detect the box location, and the blue allows it to detect the arm location.

And now the Known Issues:

- First and foremost, you might be wondering, "Why does the arm pick up the provided boxes so slowly???" and that is a great question. That really boils down to an issue with the arm itself. As you might notice going through the code, all commands have large delays hardcoded between them. This means it makes very small movements with large delays between them. This is because if you make many movement commands at once, they override each other. The commands are immediately passed to the arm, which does not buffer command inputs. This means if you send 3 movement commands at once, the arm will not move the way you intended it to. I was told that there is some kind of event system with either ROS or the arm itself that can be used to execute another command as soon as the first has finished, but I never learned how to do it, nor do I know exactly where to look for a solution.
- Next, there is an issue with the image detection. Glare screws with this thing horribly. I am investigating possible solutions and if I find one it will be listed at the bottom of the README.md on the GitHub page for the project. If not, then here are the basics of the problem: The camera cannot differentiate between bright colors and glare from reflections. Basically, if the lights are on above the arm, then there is a chance it will not work. I recommend if the arm misses the cube, try the color tester on the GUI and see what extra stuff is being picked up.
- Also, camera simply detects the centroid of whatever is being filtered. So, if it is correctly detecting the cube, then it gives the centroid of the cube. If it is correctly detecting the arm location (From the tape) then it finds the center of both pieces of tape together, which is between the limbs of the grabber. This means that if there are extra parts of the image making it through the color filter (Like glare) then the centroid location includes

the cube and the glare which means the centroid is wrong. This also means that more than one cube will screw up the centroid.

- Lastly, the color tuning done in the GUI does not actually save to the arm. Once you determine the color arrays that work for you, then you will need to edit the code to replace the color arrays as specified in my README file.

Custom Code

Now, if you want to create your own custom code to run, there are two ways of running custom functions in my code, and each has strengths and weaknesses. Either way, I'd recommend first following my TKInter code in the GUI file and making a new button. Then, the first way is to edit an existing file (Like box_pickup) to include another flag to tell them arm when a new button has been pressed and execute your custom function under that. This does rely on carefully managing flags and callbacks to ensure your code is only run at the correct time. The other way is to create a new file that executes your custom code and call that directly using subprocess from the GUI. Follow my example in the GUI code to create a subprocess. This is slightly more difficult up front, but it means you can close the file using the functions I wrote so the code only really fires when it is meant to.

Starting with the arm

To start using the arm, a good project to get the hang of the basics of using the armcam code is to make what I call the robot equivalent of a "Hello World" project, in which you will make the arm wave hello.

To start this project, you should first create a new publisher in the GUI file. Then, a subscriber to catch your publisher in either a custom file or one of the existing files.

Next, create a new button on the GUI that publishes a message from the publisher. This is the button that will start your custom program.

Now create a callback method in the same file as your subscriber. Then, in that callback function, send the arm to the home position. Next, use the publishing methods I used in the GUI to send a coordinate, and send coordinates to move the arm back and forth like the arm is waving. Make sure to put delays after each movement. Finally, have the arm return to the sleep position when it is done.

This should help you get the hang of using the arm and the weird quirks that come with controlling the arm.

Good luck using the arm, and remember: if you have any issues, google it. If you can't find it on google or if you have a question about my code, feel free to email me at benjaminblinder@brandeis.edu, or at my personal email at btblinder1@gmail.com.

From,
Ben Blinder