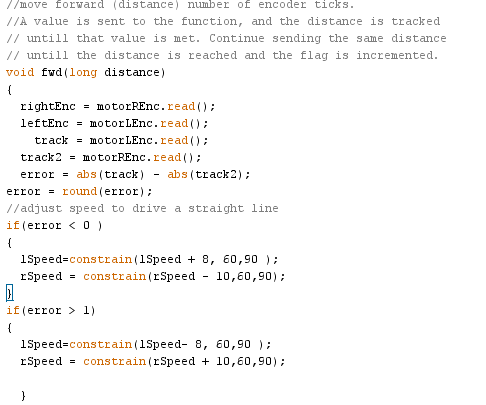
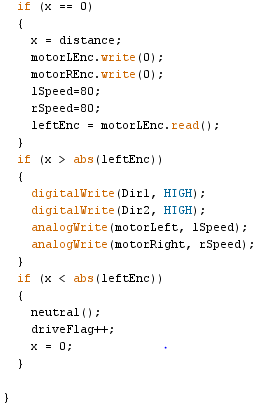
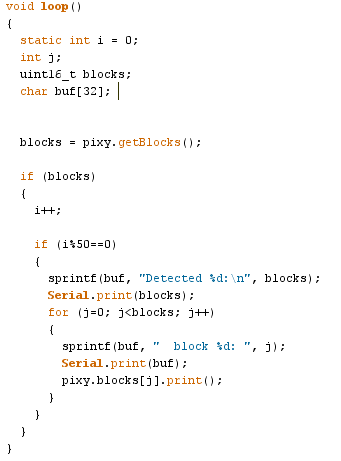
**Dive functions**

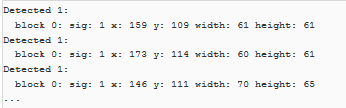
The forward function drives the robot forward until another command is given for it to stop. All pins are named variables that are initialized at the start of the program to make it easier to change pins. The function starts by setting the direction pins on the motor controller to HIGH, which moves the robot forward. It then writes a speed to the motor controller drive pins. The speed value sent is a PWM(Pulse Width Modulation) value between 0 and 255, which is a way to tell the motor controller to repeatedly turn on and off power to the motors. This results in a different voltages being applied to the motors for different values of PWM, changing the speed of the motors. In practice, values below 60 are too low to drive the motors and values above 150 tend to be too fast to control. This function is used when the robot needs to move forward until the camera indicates the robot has reached a certain distance to the robot.

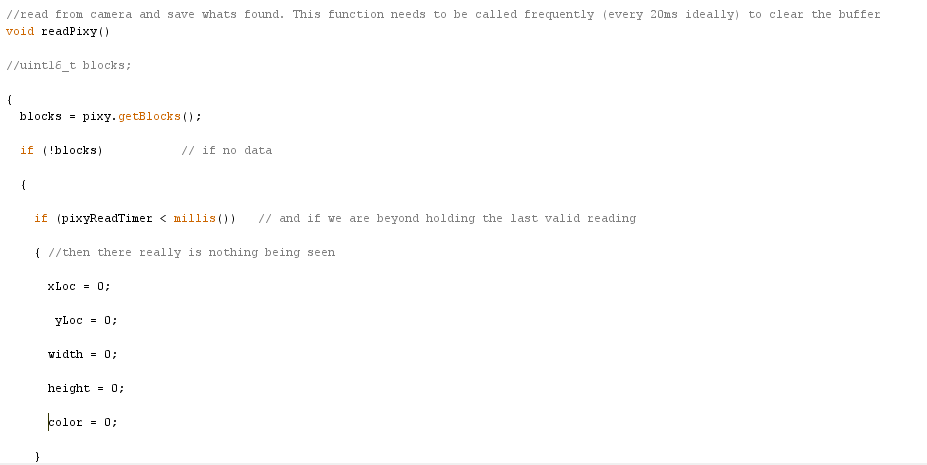
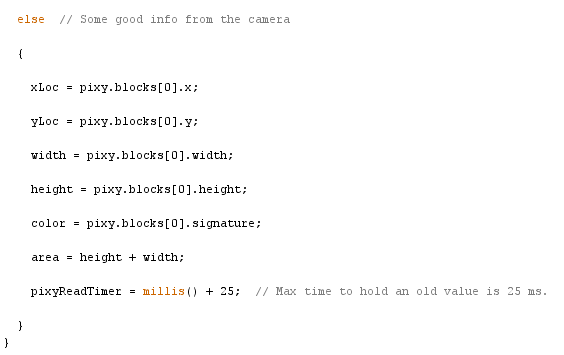
A second function, named “fwd”, was added so the robot could drive forward a set distance while attempting to move in as straight a line as possible. The biggest difference between this function and the one named “forward” is that this function receives a distance value. This value is used to determine how many encoder steps the robot should go. The goal of this function is to be able to move the robot forward, in a straight line, while not requiring the program to stay in the function. To achieve this, it first starts by reading the encoders to determine how far the robot has moved and if the robot has been moving in a straight line. If the encoder values have a difference between them, the motor speeds are adjusted to help compensate. The change in motor speeds are restricted so that one motor doesn’t turn off while another is going very fast in the event there is a problem reading the encoders. The function then checks if the variable ‘x’ has a value or not. This value is the number of encoder steps the robot needs to take. If this value is 0, then ‘x’ will be the distance passed to the function, the encoder values will be cleared, and the motor speeds will be reset to a starting speed. The function then compares ‘x’ to the values read off the encoders, and if the encoders are less than the desired distance(‘x’), the function adjusts the speed as previously determined. If the desired distance has been reached, the function will stop the robot, increment a drive flag to indicate to the rest of the program the distance has been reached, and resets the distance tracking variable ‘x’. The advantage to this method is that as long as the same value is written to the function, and the flag is used to track when the distance has been reached, the program can continue cycling through other functions to gather data or perform other actions.

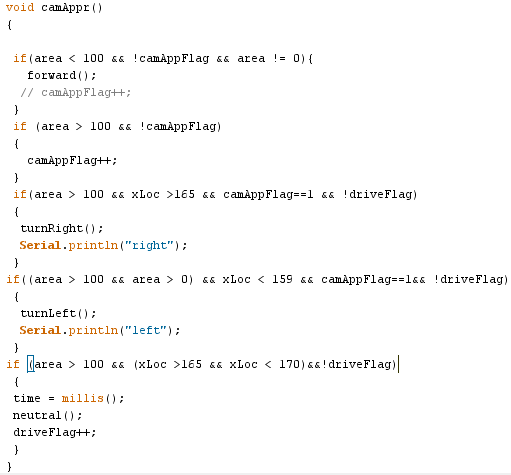
The program uses other functions based on the same principle of the **forward** and **fwd** functions, named **reverse, rvrs, turn, turnLeft and turnRight**. The only changes will be the directions, a LOW to both will drive the robot in reverse, a LOW to one direction and a HIGH to the other will cause the robot to turn.

**PIXY Camera**

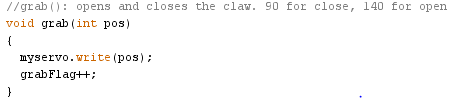
The manufacturers default code made for the pixy was useful for testing what the pixy sees. It starts by initializing variables to track object count, create a buffer of the object data and monitor the number of times the data is printed to the user. It then grabs data from the camera and checks if the camera sent any data, and if it has the Arduino will print the information to the user. It will only print data every 50 times the camera has found object so as not to overwhelm the Arduino processor. The program takes about 20 milliseconds to run, which results in object information being displayed approximately once per second. The benefit of this program in testing was to identify what the camera was seeing, and what data the Arduino was using to act on.

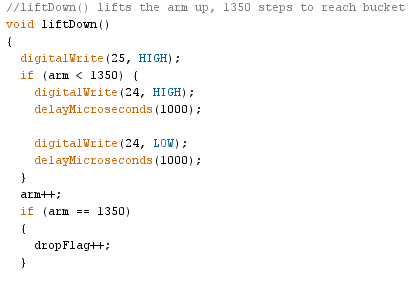


In operation, the pixy requires some tricky coding to be useful. Due to the way the manufacturer wrote the library for the Arduino, the camera will save data till the camera is read. This means the camera needs to be constantly read, and once the data is read, the cameras data is cleared, and will return a zero if called in less than 20 milliseconds because it hasn’t been updated. To work around this, the program will cycle through all functions repeatedly, and the **readPixy** function will only clear data if the camera returns no data twice in 25 milliseconds. If an object is found in the camera, the function will update variables for the color, location and size of the object found.

After the camera has been read, the data needs to be used. This is where the **camAppr** function comes in. When this function is called, the robot will drive towards the fish until the camera indicates the object is a particular size, which means the robot is close enough to attempt pick up. Once the robot is close enough, the function will increment a flag to make sure the robot does not attempt to go any closer. After this flag is raised, robot will then turn to orient itself in such a way that the fish can be picked up. When the robot is in position, the **camAppr** function will raise another flag to let the robot know its time to pick up.

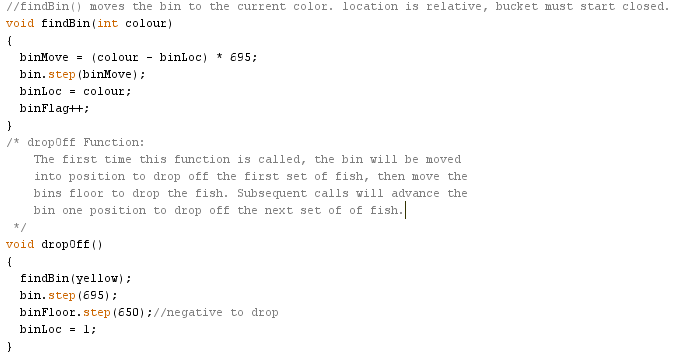
**Claw Operation**

The claw is operated by a servo, and the Arduino has built in servo libraries. The control of the claw is simply to write a position (0-180) to the servo object. Based on the way the servo is attached to the claw, sending the servo a position of 90 will close the claw, and a position of 140 will open the claw. After the claw is opened or closed, the **grab** function will increment a flag to indicate the next step in the process can be done.

**Arm Operations**

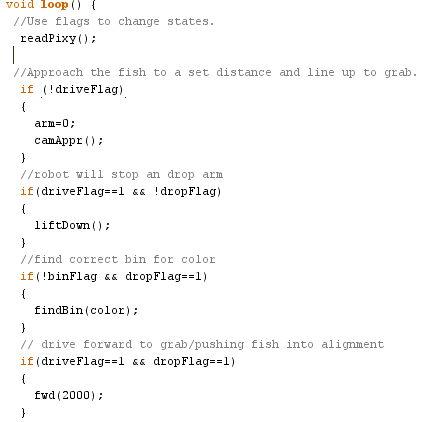
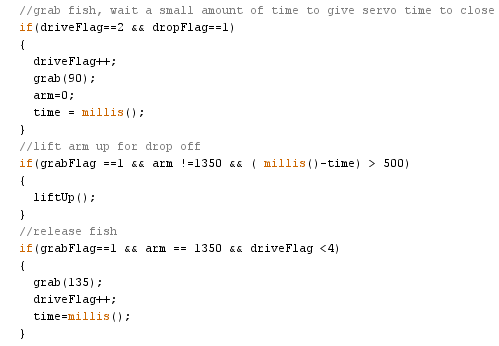
The arm is operated through the **liftUp** and **liftDown** functions. First, the Arduino pin attached to the DIR pin of the motor controller is brought HIGH to lift the arm up towards the bucket, or LOW to drop the arm down for pickup. Then the pin on the Arduino attached to the ENABLE pin on the motor controller is cycled through a HIGH/LOW state with a short delay in between (1000 micro seconds). This will move the arm through one “step”, which is approximately 1.7 degrees. The “arm” variable is then incremented to track the location of the arm. The main program will monitor the location of the arm, and only call the function as many times as needed to reach the desired location(approximately 1350 steps for full rotation).

**Bucket Operation**



The bucket is controlled by two functions. **findBin** receives the color of the fish picked up, and then determines how far to move the bucket. The stepper motor controlling the bucket uses the built in Arduino library **STEPPER** to move the bucket, which means the stepper will be given a set number of steps to move, and then the library handles the actual movement. Control is lost until the stepper moves through all the setps. The only indication of the bucket location is based on what the program has tracked, therefore the bucket must always start in a closed position. There are 695 steps between each section of the bucket, and the function determines where to move the bucket by the difference in the current colored bin, and the desired colored bin. A negative value for this difference will result in a negative number of steps, which the stepper library will interpret as a requirement to move the stepper in reverse. After the bin is moved, the variable “binLoc” holds the current color, and a flag is incremented to indicate the bucket has moved. The floor of the bucket, which is moved for drop off, is controlled by the **dropOff** function. First, this function uses the **findBin** function to move the bucket in place for drop off, and then moves the floor to drop off one set of colored fish. Subsequent calls to this function will move the floor to the next set of colored fish.

**The LOOP that pulls it all together**

The functions do most of the heavy lifting. Due to the fact the camera needs to be constantly read, the flags from each function tell the main program what its next step should be. All the functions either take a value to determine when that function has fulfilled its function. At the start, the program will read the camera. It will then go through a series of “if” statements to check where in the operation the program is, and where it needs to be. The first check is if the robot is close enough to the fish for pick up. Then it checks if the robot is close enough, and if the arm has been dropped down. After the arm is dropped down, the program checks the bin location and then checks if the robot is in position to grab the fish. The next part checks if the fish has been grabbed, and will track the time since the fish has been grabbed to delay the arm from lifting up before it has a fish. The “delay” function is not used here because it would cause the cameras buffer to build up. The final checks determine when the arm and the claw are in position to drop the fish off in the bucket.