Report:

Task 1: Explain color\_trackig\_node.cpp

**Function: cv\_process\_img**

1. Two variables are being passed into the function: input\_img and output\_img
2. The input image then gets converted to gray scale
   * Reason: It is easy to process, because you are only dealing with one layer (0 .. 255) vs the RGB which has three
3. Three variables are defined for the Canny function
   * T1 = first threshold … lower threshold
   * T2 = second threshold .. Upper threshold; typically a ratio of the first threshold recommend is
     + 2:1 and 3:1 of lower threshold
   * Next is the kernel size
4. The canny function
   * Detects the Edges of the object. Generates a mask; Our bright lines running on the black screen.

**Function: cv\_publish\_img**

1. ROS specific .. publishes sensor data

**Function: cv\_color\_tracking**

1. Zeros
   1. MatLab style initializer fills input\_image.size() with three channel 8-bit unsigned integer of zeros and initialized imgLines
2. cvtColor
   1. Takes the input\_img and converts it Hue, Saturation, Value (HSV) and stores it to imgHSV variable. This will make it easier to identify certain colors
3. inRange 1
   1. takes the imgHSV converted image searches for lower blue values stores frame threshold in mask1
4. inRange 2
   1. takes the imgHSV converted image searches for upper blue values stores frame threshold in mask2
5. Bitwise operation on the two blue masks covers the entire spectrum of Blue this is stored in mask
6. Erode and dilate functions
   1. Take mask and removes noise from the picture. Helping prevent detected objects we don’t want
7. findContours function
   1. retrieves contours from mask and stores the resulting Output Array of Arrays in contours and Output Array in hierarchy
      1. contours being very valuable because it is the points of the detected edges
8. first for loop – regulated by the number of points in contours vector
   1. contourArea
      1. calculates the Area of the detected object
   2. if the area is large then assume it has detected something
      1. keeping the area large helps eliminate noise
   3. if an area is detected count it
      1. if an object is detected more than 2 times or more publish a message
      2. otherwise ignore it.
9. Imshow functions
   1. Displays the video feed camera and mask

**TASK 2: Code added**

1. vector<Point> approx;
   1. Vector of Points to store polygonal points
2. vector<Moments> mu(contours.size());
   1. Vector of moments points size is based on contour vector size
3. vector<Rect> boundRect(contours.size());
   1. Vector to store points of Rectangle based on contour vector size
4. Two color Scalars are just to make it easy to change color of lines being drawn and rectangle box

Scalar color = Scalar(0,255,0);

Scalar color2 = Scalar(0,0,255);

1. mu[i] = moments( contours[i]);
   1. takes the contours points and converts them to moments points
      1. removes the dreaded commas
2. double peri = arcLength(contours[i], true);
   1. Calculates the contour perimeter and stores it in peri
3. approxPolyDP(contours[i], approx, 0.02 \* peri, true);
   1. approximates a polygonal based on contour perimeter and contour points
4. boundRect[i] = boundingRect( approx );
   1. stores the bounding rectangle points in the vector boundRect
5. cX and cY are the center point of the detected object

float cX = (mu[0].m10 / (mu[0].m00 + 1e-5));

float cY = (mu[0].m01 / (mu[0].m00 + 1e-5));

1. These line functions draws the X in the middle of the detected object

line(input\_img, Point2f((cX - 10), (cY - 10)), Point2f((cX + 10), (cY + 10)), color, 2);

line(input\_img, Point2f((cX + 10), (cY - 10)), Point2f((cX - 10), (cY + 10)), color, 2);

1. Determines the location of the object in relation to the center X value of the frame. The Center value of the video feed is 640. If we are between the values of 600 and 680 we assume the object is in front of us. If we are 600 or less we are left of the object, and if we are 680 or more than we are right of the object.

if (cX <= 600){

putText(input\_img, "Left", Point(900,715), FONT\_HERSHEY\_DUPLEX, 1, color2, 2);

} else if (cX >= 680) {

putText(input\_img, "Right", Point(900,715), FONT\_HERSHEY\_DUPLEX, 1, color2, 2);

} else {

putText(input\_img, "Front", Point(900,715), FONT\_HERSHEY\_DUPLEX, 1, color2, 2);

}

rectangle( input\_img, boundRect[i].tl(), boundRect[i].br(), color, 2, 8, 0 );

}