# 10.KCF object tracking

## 1. Program function description

After the program is started, select the object to be tracked with the mouse, press the space bar, and Muto enters tracking mode. Muto will maintain a distance of 1 meter from the tracked object and always ensure that the tracked object remains in the center of the screen. Press the R key to enter the selection mode, and press the Q key to exit the program. After the controller program is started, you can also pause/continue tracking through the R2 key on the controller.

## 2. Code reference path

After entering the docker container, the source code of this function is located at

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_kcftracker/src/KCF_Tracker
.cpp
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_kcftracker/launch/kcf_trac
ker_launch.py
```

## 3. Program startup

### 3.1. Start command

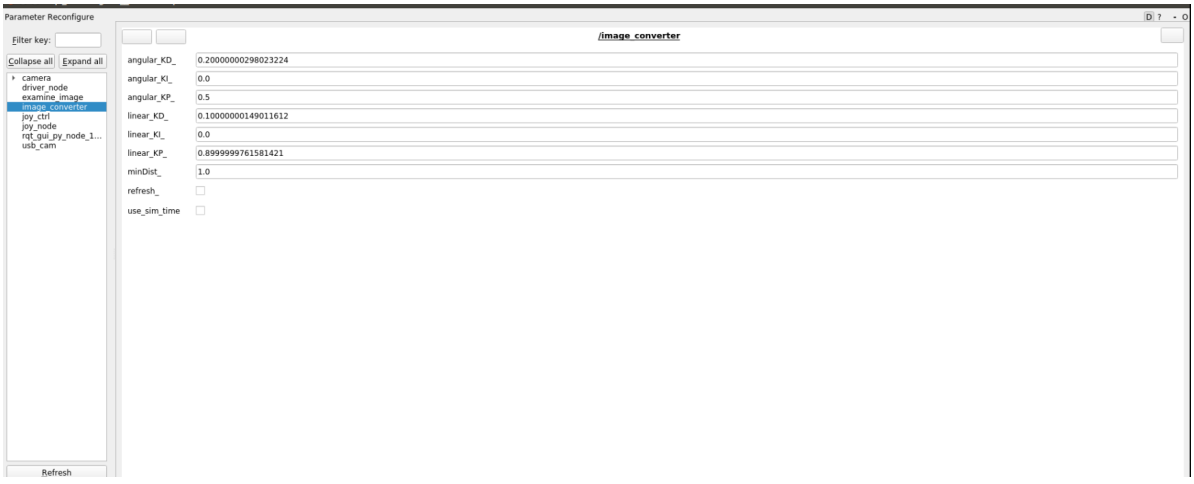
After entering the docker container, enter in the terminal,

```
#Start the depth camera
ros2 launch astra_camera astro_pro_plus.launch.xml
#Start object tracking node
ros2 launch yahboomcar_kcftracker kcf_tracker_launch.py
```


Use the mouse to frame the object to be tracked, and release it to select the target. Then press the space bar to start tracking, move the object slowly, Muto will follow and keep a distance of 1m from the object.

You can also use the dynamic parameter adjuster to debug parameters and enter it in the Docker terminal.

```
ros2 run rqt_reconfigure rqt_reconfigure
```

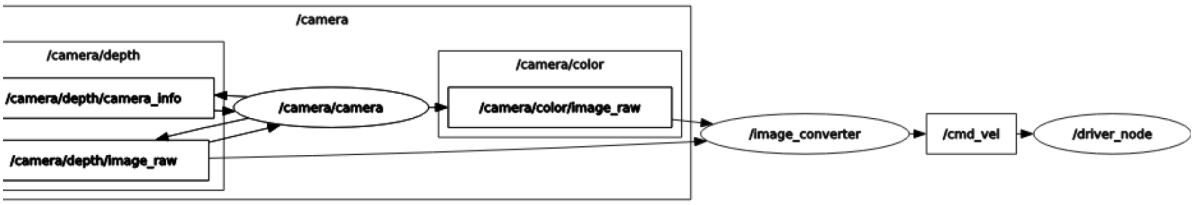



The adjustable parameters include Muto linear velocity, PID of angular velocity and tracking distance. After modifying the parameters, click "refresh" to refresh the data.

## 3.2. View node topic communication graph

You can view the topic communication between nodes through the following command,

```
ros2 run rqt_graph rqt_graph
```




## 4. Core code

The principle of function implementation is similar to that of color tracking. Linear velocity and angular velocity are calculated based on the center coordinates of the target and the depth information fed by the depth camera, and then released to the chassis. Part of the code is as follows.

```
//This part is to obtain the center coordinates after selecting the object, which
is used to calculate the angular velocity.
if (bBeginKCF) {
    result = tracker.update(rgbimage);
    rectangle(rgbimage, result, Scalar(0, 255, 255), 1, 8);
    circle(rgbimage, Point(result.x + result.width / 2, result.y + result.height
/ 2), 3, Scalar(0, 0, 255),-1);
    } else rectangle(rgbimage, selectRect, Scalar(255, 0, 0), 2, 8, 0);

//This part is to calculate the values of center_x and distance, which are used
to calculate the speed.
int center_x = (int)(result.x + result.width / 2);
int num_depth_points = 5;
for (int i = 0; i < 5; i++) {
    if (dist_val[i] > 0.4 && dist_val[i] < 10.0) distance += dist_val[i];
    else num_depth_points--;
}
distance /= num_depth_points;

//Calculate linear and angular velocity
if (num_depth_points != 0) {
    std::cout<<"minDist: "<<minDist<<std::endl;
    if (abs(distance - this->minDist) < 0.1) linear_speed = 0;
    else linear_speed = -linear_PID->compute(this->minDist, distance);//-
linear_PID->compute(minDist, distance)
}
rotation_speed = angular_PID->compute(320 / 100.0, center_x /
100.0);//angular_PID->compute(320 / 100.0, center_x / 100.0)
```