Special Topics in Statistical Pattern Recognition Final Project Report

Feiyang Cai, Weihan Wang
Vanderbilt University April 30, 2018

***Abstract*—Moving object detection and tracking is an evolving research field. There are many applications and products that using this technique, such as traffic surveillance, motion analysis, activity recognition, logistics, etc. What's more, video surveillance systems have been developing rapidly recently. Video surveillance systems detect the target in initial stage and then perform the functions like object classification, its behavioural tracking. Therefore, it is worth to implement moving object detection, tracking and following using a stereo camera on an autonomous vehicle. However due to the real-time object in dynamic tracking environment, there are many problems associated with object detection and tracking, such as illumination changes, fake motion, image noise etc. To overcome such problems, a system based on digital image processing algorithm and control algorithm in ROS is designed in this project. In addition, an attempt has been made to develop effective object detection and tracking system.**

I. Introduction

The process of locating the moving object in sequence of frames is known as tracking. This tracking operation can be performed by using the feature extraction of objects and detecting the objects in sequence of frames. Moving object detection and tracking in video sequences is an important aspect of computer vision[1].

In our project, the system consists of a Zed 2K Stereo Camera mounted on an autonomous vehicle(Traxaas F1/10th Car Platform) for tracking a moving object. Both video processing, image processing algorithm and control algorithm are embedded in Nvidia TK1 board. Autonomous vehicle is controlled by PID control algorithm via motor and servo motor. And under our system helps in accurate object detection and tracking.

This paper is arranged as follows. Section Robot Operating System and Software Structure includes hardware introduction, software structure. Section Algorithms and Implementation includes algorithm of the our tracking system i.e. video, image and control algorithm and core code. The experimental results of proposed algorithm and evaluation are presented in section Related work. Finally conclusions, limitation and improvement are drawn in final section.

II. Hardware

The Traxxas 1/10 RC car is shown in Figure 1. The ESC and the brushless DC motor are the power components for the car, which has outstanding performance and the car can reach to 60 mph. Also, we need a extra battery to provide enough power and current for the motor. The servo motor is used to control the steering of the vehicle. The servo and the motor are controlled by Pulse Width Modulation (PWM) generated by Arduino.

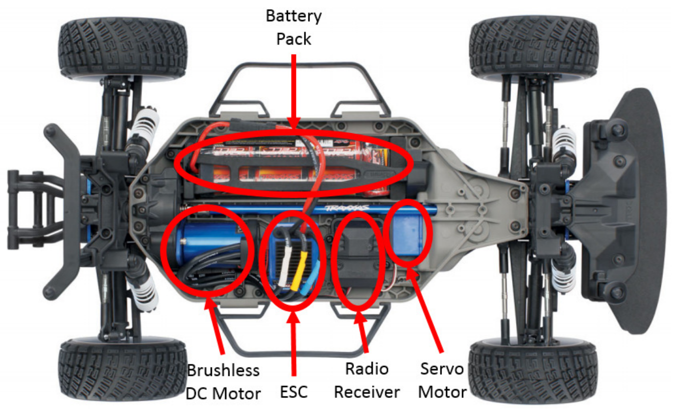

Fig. 1: Traxxas 1/10 Model

The core of the object detection is the camera, and it is shown in Figure 2. Even if the ZED camera is very powerful, which can do the depth sensing and motion tracking, we still just use it as a simple camera to realize our project object.

Nvidia Jetson TK1 is the core of the testbed, which contains a 32-bit ARM CPU and a NVIDIA Kepler GPU. The Linux Operating System runs in this board, in which we can install the ROS to control the car. Also, the sensors and network station should connect to the board for perception and communication.

Fig. 2: ZED 2K Stereo Camera

Since the Jetson TK1 cannot generate the PWM to control the servo and motor directly, the Arduino board is connected to receive the command from the TK1 and control the servo and motor directly by PWM. The whole structure of the hardware is shown in Figure 3.

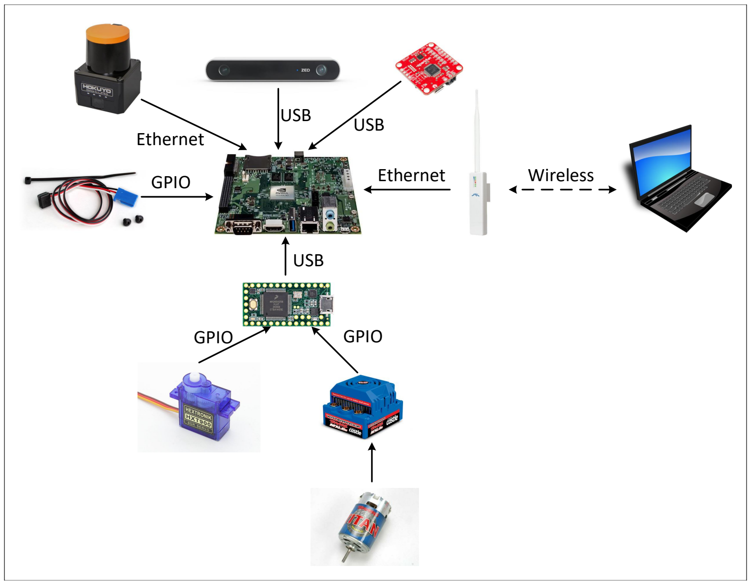

Fig. 3: Hardware Structure

III. Robot Operating System and Software Structure

Robot Operating System (ROS)[2] is a flexible framework for writing robot software, which is a collection of software used for developing code for robotic applications and providing functions which are similar to that of an operating system (OS).

The basic unit of the ROS is called nodes, which are programs and scripts written by Python or C++. Normally, one node has a specific function, such as reading the data from the sensor, executing PID control, etc.

The nodes are communicating with each other over a ROS topic. For example, one node reads the data from a sensor and publishes the message to a topic. Then, if one node need to get the data from the sensor, it should subscribe the topic to receive the data message. It should be noticed that one node can publish multiple topics also multiple nodes can subscribe the same topic. The structure of ROS node is shown in Figure 4.

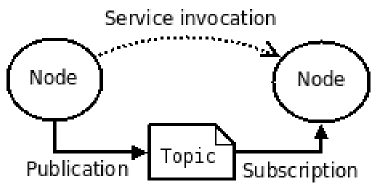

Fig. 4: ROS Node

As described above, we can perform our task based on the ROS platform. The software structure is shown in Figure 5. There are three parts of the software, one is the sensors part, another is control part in NVIDIA GPU board, and the last one is the firmware in Arduino board.

The control part is the core of the software, which controls the car detecting and tracking the object.

1) Track

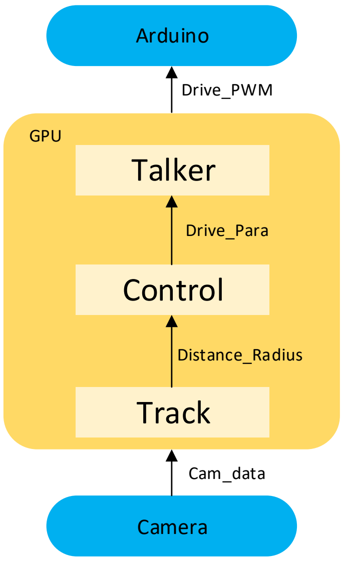

Fig. 5: Software Structure

In the Track node, it subscribes the Cam_data topic, which is the data from the camera. This node detect the object, the algorithm of the detection will be introduced later. This node calculate the distance from the center of the car to the object and publish the Distance_Radius topic.

2) Control

In the control node, it subscribes the error information from the topic Distance_Radius. The PID controller is used for the direction control.

3) Talker

In this node, which maps the PID control signal to PWM duty cycle signal and sends it to Arduino board by topic Drive_PWM.

The Arduino board controls the servo and motor directly by changing the PWM duty cycle as described above. There are one topic subscribed by the Arduino as the UART interrupt format since there is no ROS or OS running in the board.

The topic is the ”drive_PWM”, which contains the steering control and the speed control information. Since the control data is from -100 to 100, which is not the PWM duty cycle, one need to map the control value to PWM duty cycle.

IV. Algorithms and Implementation

A. PID Controller

We use the standard PID equation, as

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\mathrm{d}\tau + K_d \frac{\mathrm{d}e(t)}{\mathrm{d}t} \tag{1}$$

Due to the codes being executed on the embedded system — the GPU board, which should be discrete, then Equation (1) is simplified to discrete PID equation. Also, it is normally using only PD in steering control. The equation is[3]:

$$\Delta\varphi = K_p Error + K_d(Error - Error_1) \tag{2}$$

Where the K_p is the proportional coefficient, K_d is the derivative coefficient, $Error$ is the current error from the desired trajectory, and $Error_1$ is the error from last time. The output $\Delta\varphi$ is steering angle of the front wheels. Also, in order to prevent the angle from becoming too big, which would not match the actual situation, one gives a limitation of the output angle, from $-45°$ to $45°$.

Using PID controller we can get the output — steering angle $\Delta\varphi$, and map it to the PWM duty cycle in the Arduino, which can control the steering angle directly.

B. Gaussian Blur

In order to reduce noise, we performed a smoothing operation called Gaussian Filter which applied a filter to our image. Gaussian filtering is done by convolving each point in the input array with a Gaussian kernel and then summing them all to produce the output array.

In Gassian blur, We should specify the width and height of the kernel which should be positive and odd. In addition, We had to specify the standard deviation in the x and y directions, σ_x and σ_y respectively. If only σ_x is specified, σ_y is taken as equal to σ_x. If both are given as zeros, they are calculated from the kernel size. Gaussian filtering is highly effective in removing Gaussian noise from the image.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

We all knowed that two mathematical method are combined to produce a third method in convolution. In image processing methods are usually called kernels.

Specifically, a Gaussian kernel is a square array of pixels where the pixel values correspond to the values of a Gaussian curve (in 2D). Furthermore, multiplying each pixel in the input image with both kernels and adding the resulting values is to get the value for the output pixel.

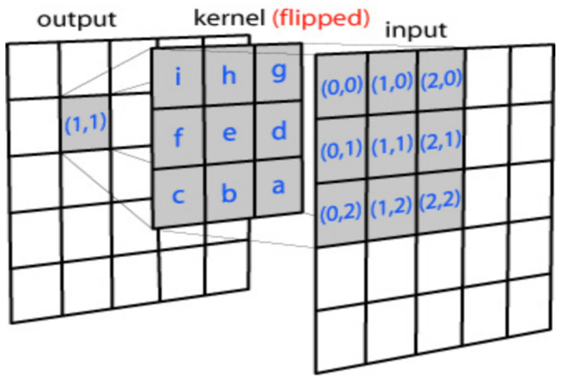

Fig. 6: Gaussian Blur Model

C. Implementation

All implement codes are running in Robot Operating System(ROS) which is embedded in Nvidia TK1 board.

We applied color detection over object detection, and to blur or smooth each frame. Meanwhile we implemented the in-built Gaussian Blurring method of OpenCV to reduce image noise and reduce detail. We used it as a pre-processing stage in computer vision algorithms in order to enhance image structures at different scales, so makes camera easier to detect colors.

Mathematically, applying a Gaussian blur to an image is the same as convolving the image with a Gaussian method that we told above.

What’more, capturing video requires capturing individual frames from the webcam running. Within the loop, each frame is processed. Each frame or each image captured is made up of pixels. These pixels consists of the 3 primary channels(RGB) color space. However it is not desirable to operate in the RGB color space since it can ignore the results based on the lighting conditions, so we chose to represent the image in HSV color space.

```
# Track object
def track():
     webcam (shitty quality)
    cap = cv2.VideoCapture(0)
    kernel = np.ones((5,5),np.uint8)
    ret = cap.set(3,640)
    ret = cap.set(4,480)
    lower_red = np.array([0, 70, 50])
    upper_red = np.array([10, 255, 255])
    pub = rospy.Publisher
    ('error', pid_input, queue_size=10)
    rospy.init_node('track', anonymous=True)
    #rate = rospy.Rate(10)
    msg = pid_input()
    while not rospy.is_shutdown():
        ret, frame = cap.read()
        frame = frame[:, :640, :]
        #Smooth the frame
        frame = cv2.GaussianBlur(
        frame,(11,11),0)
        if ret == True:
            hsv = cv2.cvtColor
            (frame, cv2.COLOR_BGR2HSV)
        else:
            continue
        mask = cv2.inRange(
        hsv,lower_red,upper_red)
        mask = cv2.erode(mask,
        kernel,iterations=2)
        mask = cv2.dilate(mask,
        kernel,iterations=2)
        mask = cv2.dilate(mask,
        kernel,iterations=2)
        mask = cv2.erode(mask,
        kernel,iterations=2)
        cnts = cv2.findContours(mask.copy()
        ,cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)[-2]
        if(len(cnts) > 0):
            c=max(cnts,key=cv2.contourArea)
            ((x,y),radius) = cv2.
            minEnclosingCircle(c)
            if radius > 5:
                cv2.circle(frame,(int(x),
                int(y)),
                int(radius),
                (0, 255, 255), 2)
                cv2.line(frame,(320,240)
                ,(int(x), int(y)),
                (0, 0, 255), 1)
                cv2.line(frame,(320,0),
                (320,480),(0,255,0),1)
                radius = int(radius)
                length = 320-(int(x))
                msg.pid_vel = 0
                msg.pid_error = length
                rospy.loginfo(msg)
                pub.publish(msg)
    cap.release()
    cv2.destroyAllWindows()
```

The PID controller has been introduced above in detail, it continuously calculates an error value $e(t)$ as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D respectively) which give the controller its name.

```
% PID control
def control(data):
    global servo_offset
    global prev_error
    global kp
    global kd
    global scale
    global pub
    global sub_bool
    sub_bool = 1
    angle = kp*error
    + kd*(error - prev_error)
    + servo_offset
    prev_error = error
    angle = trim_angle(angle)
    if check_bounds(angle):
        msg = drive_param();
        msg.velocity = data.pid_vel
        msg.angle = angle
        msg.error = error
        pub.publish(msg)
    else:
        print 'error angle
        %4.2f not in given bounds' % angle
        raise ValueError
        ('error not in given bounds')
```

Unify the velocity and angle mapping from -100 to 100. And pass real time values to arduino.

```
def callback(data):
  if stop_bool == 0:
    print'velocity=%s
    angle=%s' % (data.velocity, data.angle)
    given_vel = data.velocity
    given_ang = data.angle
    pwm_vel = mapValue(
    given_vel, -100,100,6554,13108)
    pwm_ang = mapValue(
    given_ang, -100,100,6554,13108)
    msg = drive_values()
    msg.pwm_drive = pwm_vel
    msg.pwm_angle = pwm_ang
    pub.publish(msg)

def flagCallback(msg):
  global stop_bool
  if msg.data == 1:
    pwm_vel = mapValue(
    0, -100, 100, 6554, 13108)
    pwm_ang = mapValue(
    0, -100, 100, 6554, 13108)
    new_msg = drive_values()
    new_msg.pwm_drive = pwm_vel
    new_msg.pwm_angle = pwm_ang
    pub.publish(new_msg)
    stop_bool = 1
```

Finally, the result of our project is quiet in accurate and efficient object detection and tracking.

V. Conclusion, Limitations, and Future Work

All in all, in this paper, we have proposed a system which include Zed 2K Stereo Camera, Traxaas F1/10th Car Platform, Nvidia TK1 board, and Arduino. All video processing, image processing algorithm and control algorithm are running as node in Robot Operating System (ROS) which embedded in Nvidia TK1 board, for object detection and tracking. In addition, the experimental results will be attached in Brightspace.

However, the performance of detection is not exact strictly. Therefore, in the future, we are able to apply Kalman filter which uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe to proofread detection.

References

[1] Ms NK Bhandari, Miss PotphodePratibhaM Miss NeheDipali d Miss, et al. Moving object detection and tracking. *International Journal Of Engineering And Computer Science*, 3(03), 2014.

[2] Ros official website. https://www.ros.org.

[3] F1/10 drive turorial. https://f1tenth.org.