Autonomous Vehicle + IMU, Camera and IR sensors

Yang Li, Radhen Patel, Jayakrishnan HJ, Ao Liu and Yueming Cong

***Abstract*— We describe a complete end-to-end solution to some of the challenges that defines an autonomous car. The challenges taken into consideration here are: 1) Stopping at a stop sign, 2) Avoiding a rolling ball, 3) Visual-Inertial SLAM and 4) Developing an accurate sparse map. We provide a complete list of hardware required for the project and describe how everything can be integrated. We then give a detail description of the perception and control techniques implemented for these challenges following the experiments conducted and the results obtained. The set of tools and algorithms for the challenges presented here have been integrated into the open-source Robot Operating System (ROS).**

I. INTRODUCTION

Autonomous vehicles are an increasingly hot topic in both research and applications. It is capable of tracking vehicles in the traffic stream for more accurately than a human driver can, and it can react more quickly. It doesn't have the disadvantages like getting tired, being distracted or drunk. Those would bring a lot of benefits. The first one is potential safety. According to statistics, over 35000 motor vehicle deaths in the United States in 2015. If well designed, Autonomous vehicles could also save a lot of lives. In addition to the safety benefit, Autonomous vehicles would have significant productivity increases for life and work [7]. It would save people's time by driving itself so that the driver could take a nap or focus on other work. Being able to use time spent on driving, recreation, study, and sleep would greatly improve the quality of life. Moreover, autonomous cars with autonomous driving system can make full use of cars. Think about this case, you own a car but you only drive it about 2 hours a day, going to work, shop or travel, it would waste the other 10 hours of use of it (cars might also need to rest). If we build a network of autonomous cars driving around the town, when you want to go somewhere call the nearest one. You don't even need to bother parking the car.

With all those benefits of autonomous vehicles, in this project we are trying to investigate some of the state-of-art techniques designed for autonomous driving. Basically, the car can move along the wall based on the common PID control with distance information obtained by simple IR distance sensors. One of the feature-based SLAM systems is incorporated to build a map of the environment and localize the vehicle itself simultaneously. We use a monocular camera to detect stop signs and moving balls using computer vision skills. With the detected information, the vehicle is controlled to stop in front of a stop sign and move forward avoiding the rolling ball.

In this report, we try to introduce each key hardware component used in the vehicle. The details of hardware and software integrations are followed. Driven by the chosen challenges, we talked about the techniques used to accomplish them. Then we explained how the techniques are implemented and the experiments are performed, and discussed the results. The report ends with a conclusion on our project.

II. HARDWARE

A. IR distance sensors

We used the SHARP 68 2Y0A710 F distance sensor for finding accurate position of the car from the wall. The detection range of this version is approximately 100 cm to 550 cm (4" to 32"). When interfaced with an Arduino the sensor provides distance information in cm. However, we interfaced the sensor with the servo controller itself which provided analog distance measurements. Since two such sensors were mounted on either side of the car we could take the relative distance information from both the sensors and passed through a standard PID controller to align the car between the hallway.

B. Servo controller

We used the Micro Maestro 6-Channel USB servo controller to control the servo motor for steering of the car and controlling the thrust DC motor. The controller comes with a free configuration and control program available for Windows and Linux, making it simple to configure and test the device over USB. Because the Micro Maestros channels can also be used as general-purpose digital outputs and analog inputs, they provide an easy way to read infrared sensors and control peripherals directly from a PC over USB.

C. IMU

We used the Phidgets 1042 9 DOF IMU. The IMU along with the camera was fused using an extended Kalman Filter to get a much more accurate pose estimate of the car in the map generated from SLAM. The IMU also had a ROS package which made the interface much easier.

D. Monocular camera

The camera provided in the project is a monocular camera with global shutter, called "oCam" by *withrobot* [1]. It has a variety of format including 640×480 with 80 fps. In addition, we changed the original lens to one fisheye lens with about 170-degrees field of view which, in turn, produces strong visual distortion.

[1] http://www.withrobot.com/products/cameras/

The camera is UVC compliance, so we utilized *libuvc_camera* [2] ROS package to get image from the camera.

Before using the camera, we need to calibrate the camera. The ROS package *camera_calibration* allows easy calibration of monocular camera using a checkerboard calibration target. The distortions including radial distortion and tangential distortion are brought by pinhole model [3].

Due to radial distortion, straight lines will appear curved. This distortion is represented as follows:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6).$$

Tangential distortion occurs because image taking lense is not aligned perfectly parallel to the imaging plane. So some areas in image may look nearer than expected. It is represented as below:

$$x_{distorted} = x + [xp_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy].$$

In short, we need to five parameters, known as distortion coefficients given by:

$$Distortion\, coefficients = (k_1\, k_2\, p_1\, p_2\, k_3).$$

Moreover, we also need to find information like intrinsic and extrinsic parameters of a camera. Intrinsic parameters including focal length (f_x, f_y) and optical centers (c_x, c_y) etc is called camera matrix which is expressed as a 3×3 matrix:

$$camera\, matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Extrinsic parameters corresponds to rotation and translation vectors which translates a coordinates of a 3D point to a coordinate system.

III. SYSTEM INTEGRATION

A. Hardware integration

The power system is comprised of 2 voltage levels, 5v and 7.2v. The 5v rail was used to power the IR sensors and the servo. The 7.2v rail directly powered the thrust DC motor. The camera and IMU were powered directly from the Odroid XU4 USB port. Power is supplied through two 7.2v Ni-Mh batteries.A buck converter was used to step down the 7.2v rail to 5v.The rationale behind using two batteries was to prevent the resetting of the pololu servo controller board from the magnetic inrush current from the DC motor.An electronic speed controller (ESC) was used to control the DC motor by means of PWM pulses from the maestro servo controller board.

B. Software integration

For this project, we use ROS (Robot Operating System) [4] to deal with each hardware component and integrate all using message-passing mechanism (see Fig. 1).

1) Perception: Any perception related job needs the image from the camera, one ROS node (we use libuvc_camera) needs to publish the image obtained from the camera to a topic (by default /camera/image_raw).

libuvc_camera: This package provides a ROS interface for digital cameras meeting the USB Video Class standard (UVC) using libuvc package.

Subscribed Topics

- NONE

Published Topics

- image_raw (sensor_msgs/Image) – A stream of images from the camera.
- camera_info (sensor_msgs/CameraInfo) – Camera intrinsics for images published on camera/image_raw with matching time stamps and frame IDs. If CameraInfo calibration is not available or is incompatible with the current video_mode, uncalibrated data will be provided instead.

Stop sign detection:

Subscribed Topics

- image_raw (sensor_msgs/Image) – A stream of images from the camera.

Published Topics

- stop_sign_bool (std_msgs/Int32) – A value (0 or 1) indicates whether a stop sign is detected

Rolling ball detection:

Subscribed Topics

- image_raw (sensor_msgs/Image) – A stream of images from the camera.

Published Topics

- ball_bool (std_msgs/Int32) – A value (0 or 1 or 2) indicates no ball is detected, or the ball is on the left side of the camera, or the ball is on the right side of the camera respectively.

ORB-SLAM:

Subscribed Topics

- image_raw (sensor_msgs/Image) – A stream of images from the camera.

Published Topics

- vo (nav_msgs/Odometry) – 3D pose (used by Visual Odometry) estimated by ORB-SLAM system.

IMU:

Subscribed Topics

- NONE

Published Topics

- imu_data (sensor_msgs/Imu) – 3D orientation obtained by the IMU

Robot_pose_ekf [5]: The Robot Pose EKF package is used to estimate the 3D pose of a robot, based on (partial) pose measurements coming from different sources. It uses an extended Kalman filter with a 6D model (3D position and 3D orientation) to combine measurements from wheel odometry,

[2] http://wiki.ros.org/libuvc_camera

[3] http://docs.opencv.org/3.1.0/dc/dbb/tutorial_py_calibration.html

[4] http://wiki.ros.org/

[5] http://wiki.ros.org/robot_pose_ekf

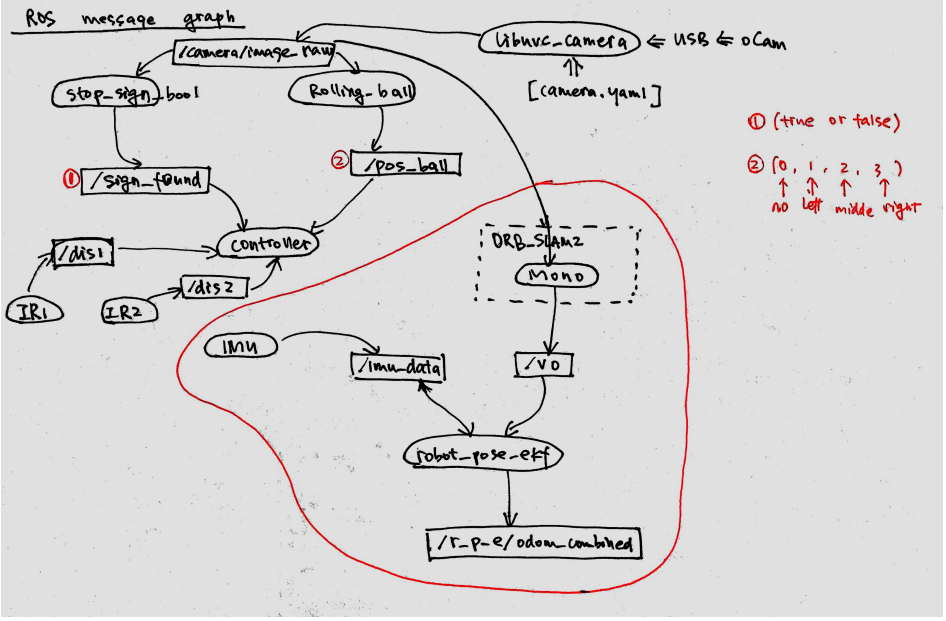

Fig. 1. ROS message graph

IMU sensor and visual odometry. The basic idea is to offer loosely coupled integration with different sensors, where sensor signals are received as ROS messages.

Subscribed Topics

- imu_data (sensor_msgs/Imu) – 3D orientation (used by the IMU): The 3D orientation provides information about the Roll, Pitch and Yaw angles of the robot base frame relative to a world reference frame. The Roll and Pitch angles are interpreted as absolute angles (because an IMU sensor has a gravity reference), and the Yaw angle is interpreted as a relative angle. A covariance matrix specifies the uncertainty on the orientation measurement. The robot pose ekf will not start when it only receives messages on this topic; it also expects messages on either the 'vo' or the 'odom' topic.
- vo (nav_msgs/Odometry) – 3D pose (used by Visual Odometry): The 3D pose represents the full position and orientation of the robot and the covariance on this pose. When a sensor only measures part of a 3D pose (e.g. the wheel odometry only measures a 2D pose), simply specify a large covariance on the parts of the 3D pose that were not actually measured.

Published Topics

- robot_pose_ekf/odom_combined (geometry_msgs/PoseWithCovarianceStamped) – The output of the filter (the estimated 3D robot pose).

IR distance sensor:

Subscribed Topics

- NONE

Published Topics

- xxx_distance (std_msgs/Int32) – A raw distance obtained by the xxx (left or right) IR sensor.

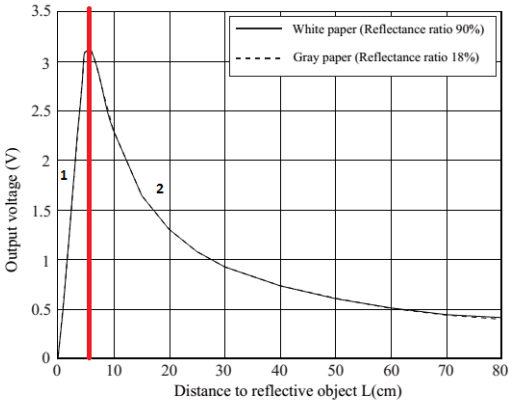

Fig. 2. IR sensor characteristics

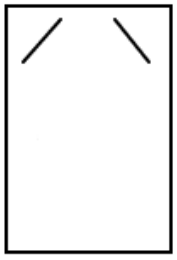

Fig. 3. Illustration of IR sensor location

2) Control: Fig. 2 represents the characteristics of the Infra red proximity sensor. The highly non linear characteristics of the sensor makes the controller design, non trivial. Also the unavailability of an accurate velocity estimate also adds to the trouble. A PID control law commands the steering of the car from the feedback of two

IR sensors placed side by side. Fig. 2 shows the IR sensor location. The sensor is also aligned at at an angle to predict corners and prepare for a turn early on. The control law is implemented as follows: The controller tries to drive the difference from the left and right IR to zero. Ideally, the car should travel equidistant from the wall edges of the corridor if the sensor had linear characteristics. But this doesn't happen in our case.The distinct advantage of this design is that even if one of the senor falls into region 1 the car wouldn't run into walls, but maintain unequal distances on either sides. As long as the corridor width is greater than 1m, at least one IR sensor would always remain in the 2nd part of the sensor characteristics. The error is given by:

$$e(t) = L(t) - R(t),$$

where $L(t)$ and $R(t)$ are the left and right IR sensor readings. The control law can be written as:

$$u = K_p * e(t) + K_d * \frac{d}{dt}e(t) + K_i \int_0^t e(t)$$

Imagine, a case where only a single IR sensor was used and a controller was designed for the 2nd part of the sensor characteristics. The car would drive in the opposite direction of the desired direction if it falls to part 1 in the middle of a run and thus run into a wall.

Subscribed Topics

- xxx_distance (std_msgs/Int32) – A raw distance obtained by the xxx (left or right) IR sensor.

Published Topics

- servo_data (ros_pololu_servo.msg/MotorCommand) – A set of data that controls steering and motor speed.

IV. PERCEPTION

A. Stop sign challenge

For the stop sign challenge, the car needs to be like a real vehicle that detects the stop sign and stops in front of it for a short period. The key point of this challenge is to detect the stop sign accurately which means the following two thing:

1) The car should detect the stop sign when there is one.
2) The car must not detect the stop sign when it doesn't exist.

We implemented two methods for feature detection: SIFT and shape-detection. Each of these has their own advantages.

SIFT is a commonly used algorithm to extract features in images. To detect the stop sign, we extract the features from the image that has only a stop sign and from the frame of the camera. Then we match the two feature sets and check if there are enough matchings. If the number of matchings if over a threshold, we consider there is a stop sign. The accuracy of the SIFT algorithm is quite good. However, the cost is too high for the Odroid. We found there is a considerable time delay during the test.

Shape-detection is a much faster method compared with SIFT. Since we only need to detect stop sign, no other things, and the shape of stop sign is octagon which is not common in the environment, we can detect the octagon instead of the real stop sign. When detecting the octagon, we transfer the image to binary and find contours in the transfered image. Then, we check for each contour if it is octagon.

B. Rolling ball challenge

For avoiding rolling ball challenge, we cannot know in advance about the specific texture of the ball, so we can not use SIFT to identify. And the object is not a fixed uniform color, so we can not use a color gray value to define, either. One possible way is using edge detection. Images of a ball taken from any angle are always round shape. That means we can detect the edge of ball by finding circles from the operator. The challenge is that after the detection accurately catch as many edges as possible, the edge point of balls should be accurately localized. We need to detect the edge with low error rate and avoid the image noises that create false edges. We should find an edge detection operator that can apply filter to smooth the image in order to remove the noise, use threshold to determine potential edges, and can get rid of the other interference items.

For avoiding rolling ball challenge, there will be a ball rolling towards our car and we need to detect and avoid it. So this challenge can be divided into two parts: ball detection and ball avoidance.

For the ball detection part, since our camera can only get the grayscale image, the most intuitive method is to detect the circle in the image. We use the Hough transform to find circles in the image and return their centers. We check those centers and tell the system if the ball is on the left side or right side of the camera image. For the ball avoidance part, if the ball is at the left half, the car is controlled to make a right turn to avoid it, otherwise a left turn.

C. Visual-inertial SLAM

In robotics, we need to map and localize the robot when it moves in an unknown environment. This kind of technique is called SLAM (Simultaneous Localization and Mapping), in which using sensor data a robot needs to build a map of the environment and meanwhile it uses the map to compute its location.

In this project, we chose visual-inertial SLAM as one challenge which fuses visual odometry information. ORB-SLAM is used to get us the visual odometry information while the inertial information is from the IMU mentioned above. The sensor data fusion is accomplished by the Extended Kalman Filter (EKF).

1) ORB-SLAM2: SLAM with a monocular camera is so called monocular SLAM. Real-time monocular SLAM becomes increasingly popular especially on two applications – navigating unmanned aerial vehicles (UAVs) and augmented and virtual reality. The major advantage of monocular SLAM while also a big challenge is the inherent scale-ambiguity. This means the scale of the world can't be observed and drifts over time since the distance information is not available from the monocular camera. Although a scaled map is not built,

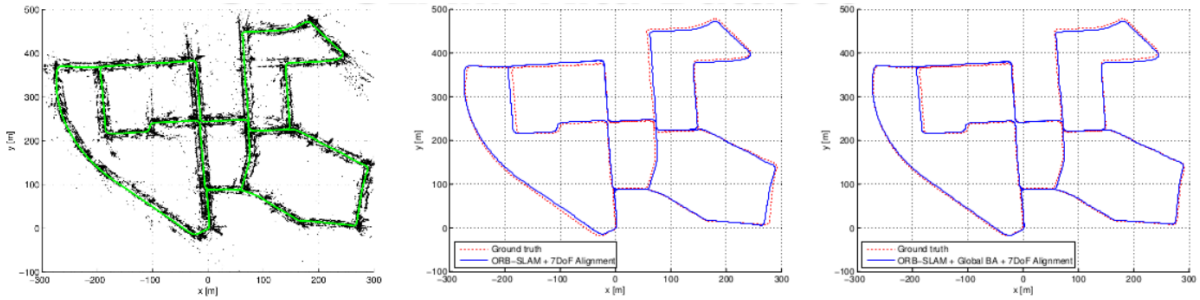

Fig. 4. ORB-SLAM on Kitti dataset, taken from [10]

this allows the system to easily switch between different scaled environments [1].

There are two main techniques that can be incorporated into SLAM: feature-based methods and direct methods. While direct visual odometry (VO) methods optimize the geometry directly on the image intensities, which enables using all information in the image, feature-based methods compute the camera position and scene geometry based on sets of feature observations extracted from images. Either has its own pros and cons, see [8], [1] for details.

Usually, we use SIFT or SURF as descriptors for detection and matching which are "expensive" as they need a lot of computation. In paper [10], the authors proposed a very fast binary descriptor based on BRIEF, called ORB, which is rotation invariant and resistant to noise.

In this project, we chose to use one of the popular feature-based SLAM called ORB-SLAM [8]. There are several reasons that pushes us to use it. The first one is that it is light-weight so that we can easily run it on the ODROID which has limited computation ability. The second one is that the authors make the implementation public on GitHub [6] and it has the ROS version since we integrates all components using ROS standard. The last one is ORB-SLAM's real-time characteristic which is the project requirement. Some recent ideas incorporated in the system are scale drift-aware loop closing [13], covisibility graph [12], and bags of binary words (DBoW) [2].

The ORB-SLAM system gets one frame from the monocular camera and then extract ORB features [10] in the frame. The system needs users interact with the camera to initialize the map by trying poses. In the map, the system stores map points, key frames, a covisibility graph and a spanning tree. After the map initialization, the system runs two threads – tracking and local mapping at the same time. Within the tracking thread, the system keep extracting ORB features from frame and estimating the current of the camera. In addition, the thread needs to update the local map and decide if the current frame is a key frame. Once there a new key frame posted from the tracking thread, in the local mapping thread it inserts the key frame to the map and then culls on recent map points and local key frames after finishing a local Bundle adjustment (BA). The other important feature of ORB-SLAM is the loop closing which composes loop detection and loop correction. In their experiment, qualitative comparisons of the trajectories and the ground truth are shown in Fig. 4. We can see the trajectories align quite well with the ground truth, especially applying global bundle adjustment.

[6]https://github.com/raulmur/ORB_SLAM2

V. EXPERIMENTS AND RESULTS

A. Stop sign challenge

We detect the octagon in the image to find the stop sign. The main steps of the detection are the following:

1) blur the image using Gaussian filter
2) transfer the image to binary
3) find all the contours using cv2.findContours()
4) check each contour if it is octagon

When testing the performance, the main issue we need to solve is noises. The detector would get a lot of contours that are considered as octagons. That is because during the octagon checking, we use *cv2.approxPolyDP()* to approximate the contour and check if the approximated contour has 8 end points. In this way, the noise contours which have 8 end points would been seen as octagons. We found most of those noises are quite small, only a few of them are big, so we use *cv2.contourArea()* to find out the contours' areas and set a range, we consider a contour as octagon only if it has 8 end points and its area is in that range. Adding constraints for area could remove a lot of noises but the range is not stable. The range varies when the enviroment changes. It is hard to find a range appropriate for all the situations.

Then we figured out another way to remove noises. First, we still set a limit. Only the contours whose area are above that limit will be considered as octagon. Then we use *cv2.isContourConvex()* to check if the contour is totally convex. And the performance of this method is much better since we have removed the small noises and the shape of the large noises are random. There is little chance for the random contours to be a totally convex shapes. However, the shape of the stop sign is very regular and it is easily to be detected as convex.

Fig. 5 shows our testing results and corresponding binary images for the stop sign detector with different distances from the camera and the stop sign. The binary shows quite perfect octagons. We can see that the detector only detects the stop sign and there is no noise. When there is no stop sign, the detector won't detect anything.

Fig. 5. Stop sign detection. Test the stop sign detector with different distances from camera to the stop sign. The stop sign is hold vertically and the light cast on the sign evenly. The binary images show the nearly perfect octagons.

Fig. 6. Stop sign detection error case. Stop sign is hold with an angle facing upwards. From the binary image, we can see that the contour of the octagon has been damaged since the light cast on it is not even.

One problem for this challenge is the light. Since the camera provided can only get grayscale image and stop sign detector need to transfer images to binary. The light need to cast evenly on the stop sign. The other one is the perspective of camera, Fig. 6 shows that when we holds the stop sign with an angle, the light casted on it is not even and the binary image shows that the shape of the stop sign is damaged by this problem.

B. Rolling ball challenge

We use *cv2.HoughCircles()* to detect the circles in the grayscale image. Canny edge detector is called to extract edges from the grayscale image, then the circle detection for the edges is executed. There are a few things that affect the performance:

1) blur methods
2) edge_thres: threshold for canny edge detector
3) circle_thres: threshold for circles detection

Blurring the image one commonly used method to decrease the noise in a image. We have tried the *GaussianBlur* and the *MedianBlur*, and their performance are close. In our system, we chose to use *GaussianBlur* method.

Threshold for Canny edge detector decides how many edges can be extracted from the image and threshold for circles detection decides what kinds of edges can be seen as circles. So there will be four ways to tune these parameter:

1) increase edge_thres and increase circle_thres
2) increase edge_thres and decrease circle_thres
3) decrease edge_thres and increase circle_thres
4) decrease edge_thres and decrease circle_thres

The first one will result in less edges and even harder to detect circles. The fourth one will result in more edges and also so many wrong circles so that the true ball won't be detected. And the other two are more reasonable. The problem of the second tuning method is that we can get

Fig. 7. Ball detection. The left and middle images get good results since the difference of color between the ball and the background is quite big. Their edge images show relatively perfect circles. The right image detects the tennis ball instead of the bigger one since the pattern on the bigger ball makes its edge image not a circle and the background and the ball are both white.

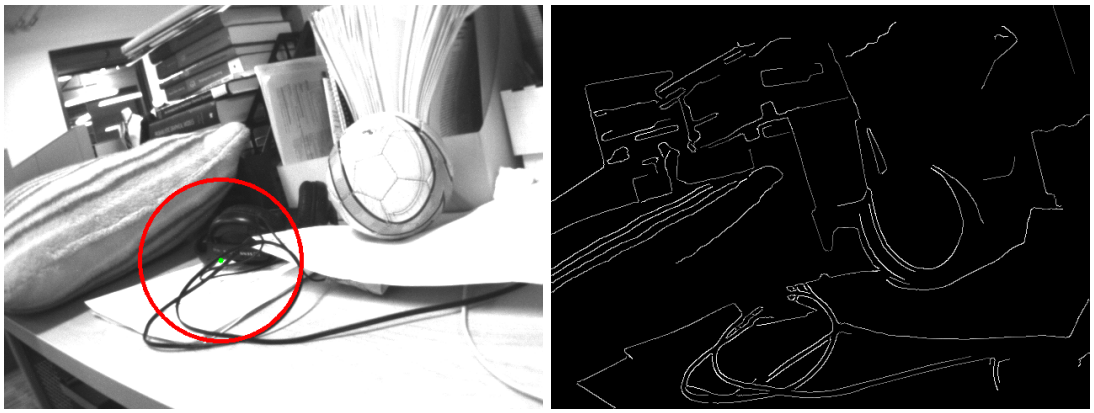

Fig. 8. Ball detection error case. The detection gets error since we only return the one which is most likely. And from the edge image, we can see the wire forms a better circle than the ball.

only a few edges, so we need to guarantee that we can at least get the outline of the ball. This is mainly affected by the color of the ball and the background. If they have the same color, the edges between them will probably not be detected. For the third method, we will get a lot of edges and some noises. More complicated the background is, more noises we will get. According to the things above, the detector works best when then background is clean and has different color from the ball's, and the ball should be also clean, since the pattern on the ball might affect the detection. It will be better to make the light cast evenly on the ball. Since the shadow will also affect the detection.

From left and middle images of Fig. 7, we can see the when the colors of the ball and the background have a big difference. The detection results are quite good. We set two ball in the right image of Fig. 7 but we only get one circle. That is because we make our detector return the most likely circle. We use it to remove a lot noises.

Fig. 8 is a test example under complex background, it shows the wrong circle because the pattern on the ball makes it more like a ellipse not a circle, and the wire is more like a circle compared with other edges.

C. Visual-inertial SLAM and sparse map

This challenge can be separated into two parts: visual SLAM and visual-inertial data fusion. As mentioned in Section IV-C, ORB-SLAM is utilized here to get visual odometry information and also a map defined with sparse feature-based points. Then visual odometry is fused with IMU data by the robot_pose_ekf package to get noise smoothed combined odometry.

In the original code, three threads including tracking, mapping and viewing are running at the same time. We need to modify the code since the whole program doesn't publish

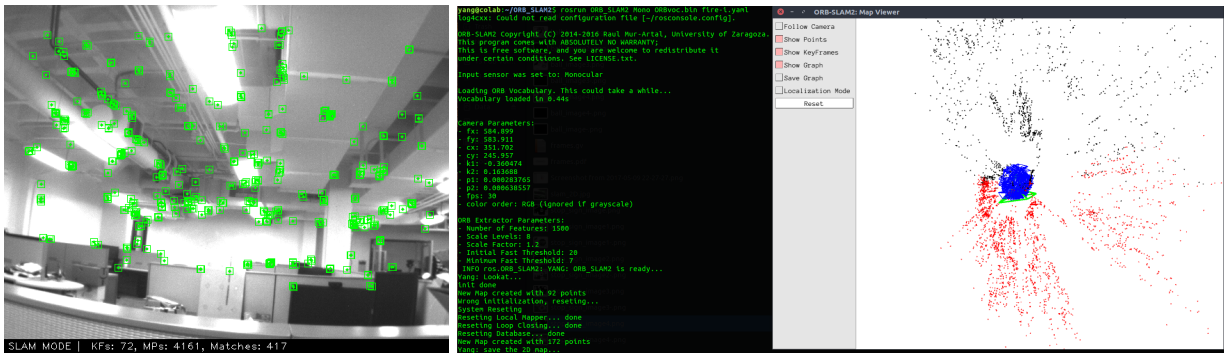

Fig. 9. ORB-SLAM around chair. Tested it with the monocular camera provided by making the camera looking around the lab space centered in my table. (Left) The current frame with the ORB featured detected; (Right) The local map built. The green triangle is the pose of camera located in the map. The blue ones are the key frames so far. Red points are features that have direct relationship with the features in the current frame, while black points are features that don't have.

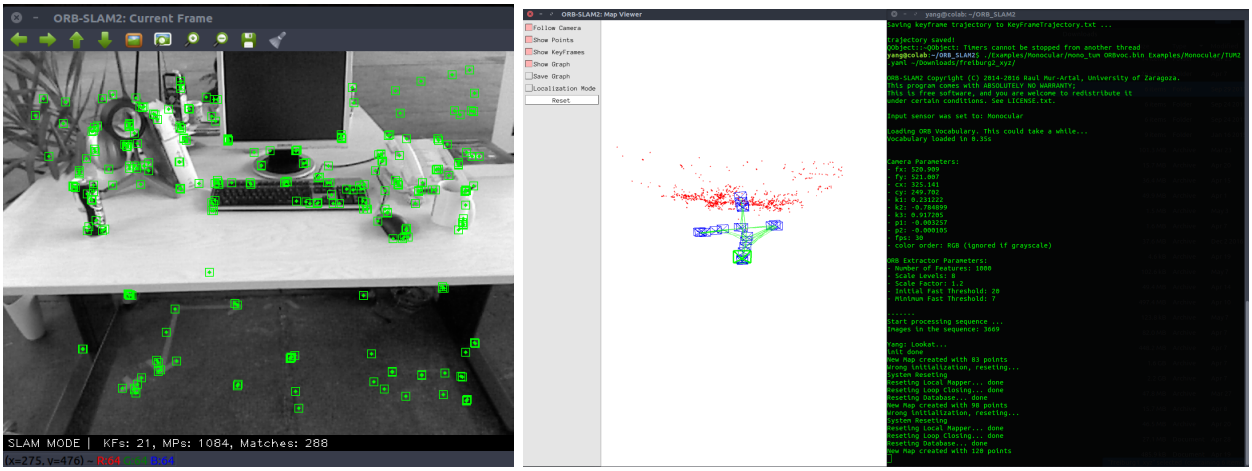

Fig. 10. ORB-SLAM tested on dataset [fr2/xyz]: (Left) The current frame with the ORB featured detected; (Right) The local map built. The green triangle is the pose of camera located in the map. The blue ones are the key frames so far. Red points are features that have direct relationship with the features in the current frame, while black points are features that don't have.

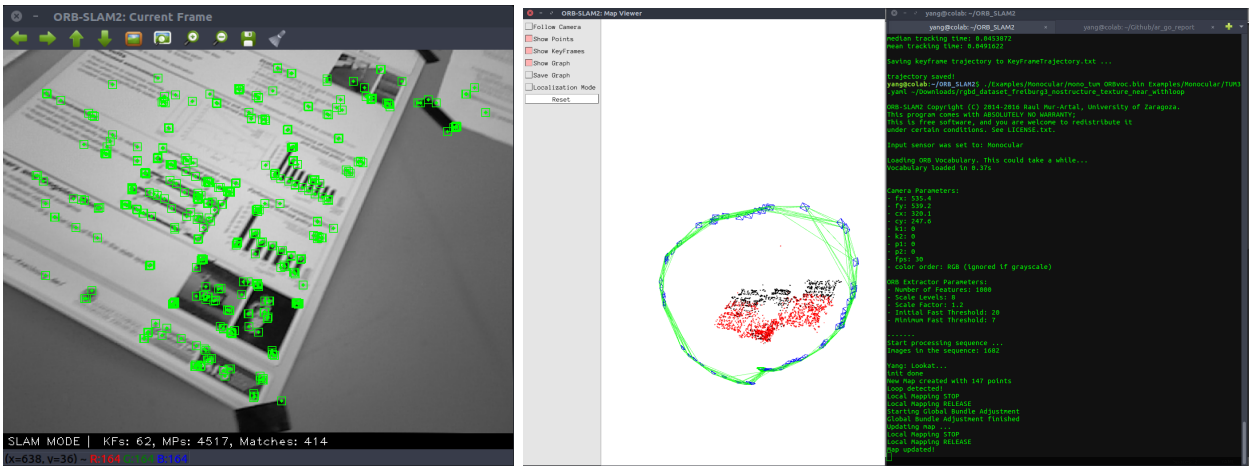

Fig. 11. ORB-SLAM dataset [fr3/nostructure_notexture_near_withloop]: (Left) The current frame with the ORB featured detected; (Right) The local map built. The green triangle is the pose of camera located in the map. The blue ones are the key frames so far. Red points are features that have direct relationship with the features in the current frame, while black points are features that don't have.

estimated pose information (scale unknown), but displays the real-time frame with ORB feature and the map with key frames, key points and the covisibility graph as shown in Figs. 9 10 11 [7].

[7]the last two datasets can be found http://vision.in.tum.de/data/datasets/rgbd-dataset/download

We tried fusing visual odometry and inertial data into EKF by publishing to /vo and /imu_data topics and run the robot_pose_ekf node at the same time (similar to what happened in course homework). However, the outputs from it didn't align with the inputs well.

DISCUSSION AND CONCLUSION

The autonomy of a self-driving car is defined at different levels. For this particular project we could successfully complete most of the challenges which defined an autonomous car. Stopping at a stop sign worked pretty well. The perform of the ESCs provided was rather unsatisfactory. It failed at providing a constant motor speed to move the car. It thus became difficult to set the PID gains to steer the car parallel to the hallway. Replacing it with a better ESC worked for us. Another problem that we encountered was to have an accurate control over the steer of the car with just one distance sensor placed on the side of the car as an input. The values of the infrared sensors kept floating around a single measurement which made it difficult to set the PID gains. Using the relative information from both the sensors on either side of the car let us have a better control over the steering. Our ball detection algorithm worked under certain conditions and so we decided not to take part in that challenge.

In our opinion, designing a better performing autonomous vehicle requires both reliable hardware components and specifically implemented and tuned algorithms. Without precise sensor data, even the best algorithms are confused and can't predict well thus making wrong decisions. On the other side, we can't make good use of data if the algorithms are "stupid". In addition, data fusion is also of importance which can make a result of $1 + 1 > 2$.

ACKNOWLEDGMENT

Thanks to Professor Christoffer Heckman for discussing ORB-SLAM, PID control, Stephen McGuire for the help with the monocular camera (libuvc_camera and robustness). We would also like to extend our gratitude to Sina Aghli who helped us debug issues with the Electronic speed control and Vikhyat and team for lending their servo motor.

PROJECT SUPPLEMENTARIES

Check our Github repo (ar_go [8]) for our meeting notes and instructions on installing components.

Check our Github repo (ar_go_ws [9]) for our codes implemented for different challenges.

Check Fig. 12 for the look of the final design of the autonomous car.

REFERENCES

[1] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[2] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

[8] https://github.com/yangautumn/ar_go

[9] https://github.com/yangautumn/ar_go_ws

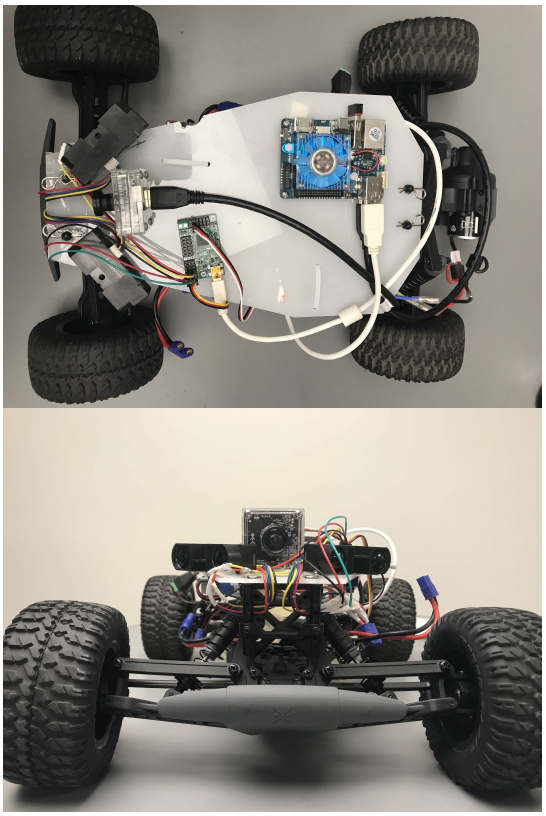

Fig. 12. Top and front views of the final design of the autonomous car

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[4] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*, 2015.

[5] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based torcs. 2013.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[7] Jerome M Lutin, Alain L Kornhauser, and Eva Lerner-Lam MASCE. The revolutionary development of self-driving vehicles and implications for the transportation engineering profession. *Institute of Transportation Engineers. ITE Journal*, 83(7):28, 2013.

[8] Montiel J. M. M. Mur-Artal, Raúl and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[9] Dean A Pomerleau. *Neural network perception for mobile robot guidance*, volume 239. Springer Science & Business Media, 2012.

[10] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[12] Hauke Strasdat, Andrew J Davison, JM Martìnez Montiel, and Kurt Konolige. Double window optimisation for constant time visual slam. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2352–2359. IEEE, 2011.

[13] Hauke Strasdat, JMM Montiel, and Andrew J Davison. Scale drift-aware large scale monocular slam. *Robotics: Science and Systems VI*, 2010.

[14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

APPENDIX

Deep Learning for an autonomous self driving car.

Steering and control of an autonomous self-driving car entails great perceptual understanding of highly dynamic environments likes highways or urban environments. Highways tend to be more predictable and orderly, in the sense the lanes are well marked and the road surfaces are typically maintained. In contrast, residential or urban driving environments feature a much higher degree of unpredictability with many generic objects, inconsistent lane-markings, and elaborate traffic flow patterns.

For the current project we used a pair of infrared sensors to steer the RC car in a hallway. In real world scenario where we want the car to drive on an open road, presumably in a straight line while dogging cars/obstacles in the front, infrared distance sensors do not seem to be the best choice. Firstly, distance sensors (infrared or sonar) need an obstacle (or a reference position) to know its location. This does not apply for highways, where there are effectively no walls on the sides, or urban areas, where there is a lot of clutter. Second, there is a limitation on the range of the sensors which limits their use in avoiding vehicles in the front. The range can be compromised in some specific scenario in avoiding obstacles in the front but for steering they are not the best choice.

There are more expensive infrared light sensors like LIDAR which give an accurate depth information of the environment. Decision can me made based off the 3D map obtained from the camera and necessary actions can be taken. But again due to their cost their use is seen less. Compared to sonar and radar, cameras generate a richer set of features at a fraction of the cost. By advancing computer vision, cameras could serve as a reliable redundant sensor for autonomous driving.

Classic computer vision techniques simply have not provided the robustness required for production grade automotives; these techniques require intensive hand engineering, road modeling, and special case handling. Considering the seemingly infinite number of specific driving situations, environments, and unexpected obstacles, the task of scaling classic computer vision to robust, human-level performance would prove monumental and is likely to be unrealistic. Deep learning, or neural networks, represents an alternative approach to computer vision. It is a data centric technique, requiring heavy computation but minimal hand-engineering. A neural network, after training for days or even weeks on a large data set, for all types of driving situations (rain, snow, night, day, etc.) can be made capable of inference in real-time.

Convolutional Neural Networks (CNNs) have had the largest success in image recognition in the past 3 years [6], [14], [3], [11]. Prior to the widespread adoption of CNNs, most pattern recognition tasks were performed using an initial stage of hand-crafted feature extraction followed by a classifier. The breakthrough of CNNs is that features are learned automatically from training examples. The CNN approach is especially powerful in image recognition tasks because the convolution operation captures the 2D nature of images.

In the past decade, significant progress has been made in autonomous driving. To date, most of these systems can be categorized into two major paradigms: mediated perception approaches and behavior reflex approaches. Mediated perception approaches parse an entire scene to make a driving decision. It involves multiple sub-components for recognizing driving-relevant objects, such as lanes, traffic signs, traffic lights, cars, pedestrians, etc. [4] uses a camera, Lidar, Radar, and GPS to built a highway data set consisting of 17 thousand image frames with vehicle bounding boxes and over 616 thousand image frames with lane annotations. They then trained on this data using a CNN architecture capable of detecting all lanes and cars in a single forward pass. Unlike other robotic tasks, driving a car only requires manipulating the direction and the speed. This final output space resides in a very low dimension, while mediated perception computes a high-dimensional world representation, possibly including redundant information. Instead of detecting a bounding box of a car and then using the bounding box to estimate the distance to the car, why not simply predict the distance to a car directly?

On the contrary, Behavior reflex approaches construct a direct mapping from the sensory input to a driving action. This idea dates back to the late 1980s when [9] used a neural network to construct a direct mapping from an image to steering angles. To learn the model, a human drives the car along the road while the system records the images and steering angles as the training data. More recently, [5] trained a large recurrent neural network using a reinforcement learning approach. The networks function is the same as [9], mapping the image directly to the steering angles, with the objective to keep the car on track. Although this idea is very elegant, it can struggle to deal with traffic and complicated driving maneuvers for several reasons. Firstly, in cases where the input images are almost the same, different human drives might have completely different decisions on what to do. When all these scenarios exist in the training data, a machine learning model will have difficulty deciding what to do given almost the same images. Secondly, the direct mapping cannot see a bigger picture of the situation.

We hence desire a representation that directly predicts the affordance like the angle of the car relative to the road, the distance to the lane markings, the distance to cars in the current and adjacent lanes for driving actions, instead of visually parsing the entire scene or blindly mapping an image to steering angles. We thus propose to learn a mapping from an image to several meaningful affordance indicators of the road situation. With this compact but meaningful affordance representation as perception output, we could demonstrate

that a very simple controller can then make driving decisions at a high level and drive the car smoothly.

To efficiently implement and test this approach we need to collect a large set of training data. TORCS (The Open Racing Car Simulator) is one of the several open source driving game which is widely used by AI researchers to collect critical indicators for driving, e.g. speed of the host car, the host cars relative position to the roads central line, the distance to the preceding cars. The collected data in the training phase can then be used to train a CNN as our direct perception model to map an image to the affordance indicators in a supervised learning manner.