Workshop on Robot Perception of Humans Baden-Baden, Germany,
June 11, 2018. In conjunction with IAS-15

A Novel User Interface for OpenPTrack based on Web Technologies

Nicola Castaman[1], Jeff Burke[2], and Emanuele Menegatti[3]

[1] IAS-Lab, Dept. of Information Engineering, University of Padova, Padova, Italy
and IT+Robotics srl, Vicenza, Italy
castaman@dei.unipd.it,
[2] REMAP in the School of Theater, Film and Television at UCLA, Los Angeles, California, USA 90095
jburke@remap.ucla.edu,
[3] IAS-Lab, Dept. of Information Engineering, University of Padova, Padova, Italy
emg@dei.unipd.it

Abstract. This paper presents a novel user interface for the OpenPTrack tracking system aimed at reducing the technical skills needed to run an OpenPTrack installation.
OpenPTrack is a ROS based system that creates a scalable, multi-camera network able to track many people and objects over large areas in real time. It is designed to support applications in education, art and culture.
The proposed solution gives a clear view of the status of an OpenPTrack network and allows to quickly recognize a fault and its cause. This user interface aims for usability, portability, and platform independence in order to be used also by non-technical users.
In detail, we used NodeJS to communicate with ROS and the underlying Linux operating system in order to collect data from the OpenPTrack network and the operating system. On the other hand, the user interface is realized as a web application using Polymer and it collects data from each node through WebSockets.
The proposed interface is currently in test both at the IAS-Lab of the University of Padova and at the REMAP of UCLA.

Keywords: OpenPTrack, people tracking, pose recognition, object tracking, web interface, web application

1 Introduction

Nowadays, there is an increasing interest in robots and intelligent systems that co-exist and collaborate with humans in their work and everyday life. A reliable perception of humans and the environment in real time is a key to realize such robots and systems. In this field, the ability to detect and track people and objects is useful for a variety of applications in the security and safety field, ranging from surveillance to robotics, as well as education, film making, and art.

OpenPTrack[4][6, 5] is an open source project launched in 2013, led by UCLA Remap and Open Perception. OpenPTrack creates a scalable, multi-camera solution able to track many people over large areas in real time exploiting some open-source libraries such as ROS[8], PCL[9] and OpenCV[2]. This system is designed to support applications in education, art and culture, as a starting point for exploring group interaction with digital environments.

In 2017, the new OpenPTrack V2 ("Gnocchi")[5] was presented. The main features of this new release are the Pose Recognition[3] and the Object Tracking[11]. In particular the Pose Recognition add the capability to annotate person tracks with poses recognized from a pre-trained set, using machine learning-based skeletal tracking.

Currently, OpenPTrack tracking data are visualizable using RViz, a 3D visualization tool integrated in ROS. Figure 1 depicts a snapshot of the OpenPTrack network in execution at the IAS-Lab, University of Padova. In detail, in this snapshot are visible people and skeletal tracking capabilities of OpenPTrack.

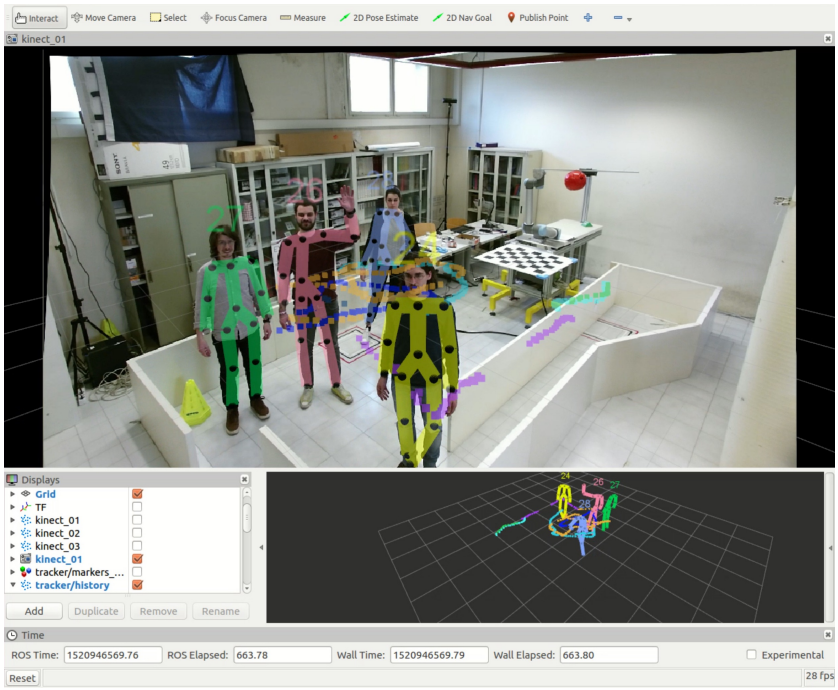

Fig. 1. OpenPTrack in execution visualized in RViz. In the snapshot is visible a camera view in which are visualized skeletal tracks and the ID assigned to each person.

[4] http://openptrack.org
[5] https://github.com/OpenPTrack/open_ptrack_v2

Even if the system is designed to be used in non-technical applications into schools or theaters, a simple user interface to monitor the status of the entire network (e.g. cameras are working, detectors are publishing) is lacking. Moreover, a network with more than 4/6 cameras is difficult to be monitored and debugged also for the developers involved in the project. Currently, the only way that developers have to get any information of the state of the network is through a lot of terminals as is visible in Figure 2. Without more information, during software testing it is sometimes difficult to discern if a problem is caused by a bug of the software or by a network fault.

This project aims to fill this lack developing a user interface that clearly visualizes the status of an OpenPTrack network. This allows also to unskilled people an easily monitoring. In future we aim also to manage (e.g. start, calibrate) an OpenPTrack network using the same interface. The proposed user interface aims for usability, portability, and platform independence. For these reasons the proposed interface is web-based so it is usable indiscriminately on Windows, macOS and Linux, as well as on tablets with Android or iOS. A web-based structure, in addition, enables new possible scenarios such as remote assistance and monitoring, and many others.

The Robot Web Tools project[1, 10] proposes a cross-platform web interface to interact with ROS. It is a client-server architecture used for developing web-accessible user interfaces for robots. This project exploits and extends the generic *rosbridge*[4] protocol. *Rosbridge* can operate over a variety of applications including web applications by transporting JSON-formatted messages over TCP sockets and WebSockets.

Even though this project is similar to what we wish to achieve for OpenPTrack, it is lacking in some crucial points. The Robot Web Tools exposes ROS but does not provide any information on the computer in which is running, neither it allows to launch a ROS node. Moreover, it creates only one socket for the entire ROS network, instead we aim to provide information on each node separately.

In the next sections the new interface is presented. In Section 2, we focus on the system architecture. In particular, Subsection 2.1 explains how the data are collected, Subsection 2.2 explains the user interface and Subsection 2.3 explains how the user interface integrates with OpenPTrack. In the end, we provide an overview of some future work that could bring this novel user interface to the next level.

2 System Design

An OpenPTrack network is composed of several RGB-D sensors, each one connected to a computer. Each one is a node called detector. There is also a master computer in the network that roles as tracker. The tracker collects the detections from each detector and merges all this information to perform a robust tracking of people or objects.

Since there could be an undefined number of detector nodes, a key point is the scalability. The user interface has to adapt to networks of different sizes and each

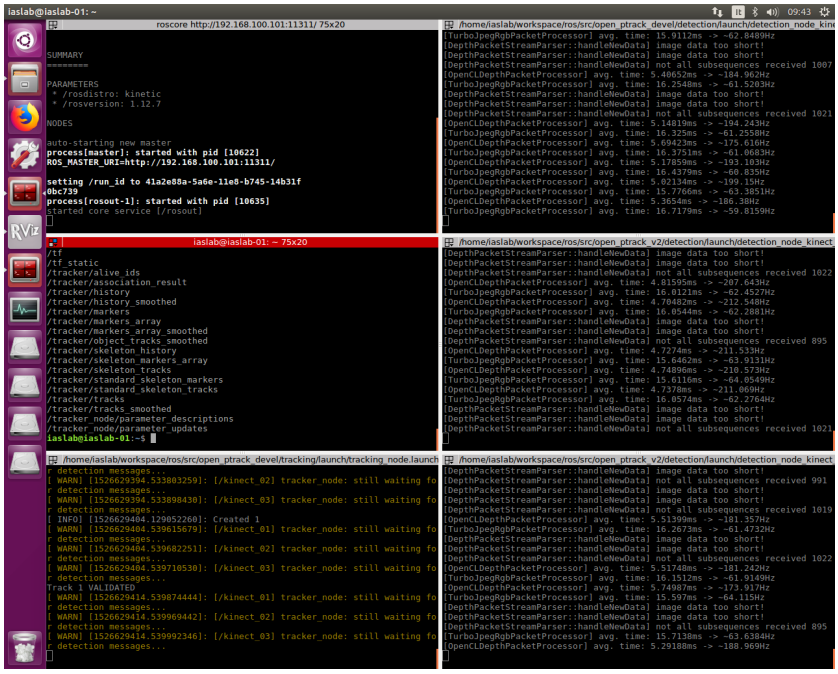

Fig. 2. Some terminals that visualize the status of an OpenPTrack network.

node has to provide its own information. To achieve this objective, each node (both detector and tracker) provides information on the status of the node. To do this, we decide to use the well-known JavaScript runtime NodeJS[6]. NodeJS uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Moreover, NodeJS is widely used to build scalable network applications.

OpenPTrack is constantly evolving with the addition of new modules and features. A modular architecture and the independence from ROS are mandatory to quickly adapt to any evolution and to remain usable also during the tests and debugs. The independence from the OpenPTrack system allows also to maintain two separate development velocities and to add new features indistinctly.

To create a platform independent user interface, it is developed as a web application. In this way, it can be used with any operating system through a browser.

2.1 Data Collection Node

The proposed user interface has to collect data from networks of different size. The adaptability of the proposed solution is therefore mandatory.

To fulfill this target, each detector runs a NodeJS node. This node collects the information from ROS and also from the underlying operating system. ROS data

[6] `https://nodejs.org/`

are acquired using the ROSNodeJS[7] library developer by Rethink Robotics and available under Apache License 2.0. ROSNodeJS is a NodeJS implementation of ROS, it provides a client library that enables NodeJS developers to quickly interface with ROS Topics, Services, and Parameters.

The Robot Web Tools, as said in the Introduction, is not able to collect information of the operating system, neither is able to operate without ROS in execution. Therefore, the new tool proposed in this paper was developed.

Each node makes available the collected data through a WebSocket created using the WS library, released under MIT License. This library was chosen instead other like the well-known Socket.IO because today the WebSocket protocol is natively supported in most major browsers allowing to exclude all the overhead introduced by the several features of Socket.IO.

Standard messages for the communication between the NodeJS node and the web interface is not defined yet. During the development phases we followed some rules for messages definition and standard messages will be defined in the future versions of the system.

2.2 Web Interface

The developed user interface has the objective to provide both an overview of the status of the entire system and detailed information on each node. In this way, the user is able to quickly detect any faults. It collects data made available by each node through a WebSocket. Obviously, the IP address of each node is memorized in the configuration file of the interface.

The web interface is developed using the Polymer[8] library. Polymer is an open-source JavaScript library for building web applications using Web Components. Web Components are a set of features that allow for the creation of reusable components in web documents and web applications.

Polymer is currently used in TensorBoard, the web application of TensorFlow, a famous library for machine learning.

Beside Polymer we took in consideration other two libraries: Angular[9] and React.js[10]. Angular is the most famous of the three libraries but is lacking on Web Components side. Moreover, Angular compared to the other two is computationally heavy and it has a high learning curve. React.js is a good competitor of Polymer, but the last one uses more open web technologies and web standards. Moreover, connection with third party libraries, such as D3.js, is very easy with Polymer.

D3.js is a JavaScript library for producing dynamic, interactive data visualizations starting from structured data. It is used inside the proposed interface to generate dynamics graphs and in future to visualize the positions of tracked people inside a map.

[7] `https://github.com/RethinkRobotics-opensource/rosnodejs`
[8] `https://www.polymer-project.org`
[9] `https://angular.io`
[10] `https://reactjs.org`

To collect data from each node the web application uses a WebSocket connection (one for each node).

WebSocket is a communications protocol that enables an interactive communication between a browser and a server. It enables an event-driven communication, facilitating real-time data transfer between client and server.

WebSocket is standardized by the W3C and is currently supported in most major browsers.

Overview Page The *Overview* page (depicted in Figure 3), as the name suggest, presents an overview of the entire system. In particular, the page shows the status of each node in the camera network, along with some other high level, but critical information such as time sync between the operating system and ROS, the number of people and objects tracked and the detection/tracking refresh rate. A list of the people and the objects tracked are also visualized. In a future update we aim to add also a map that show the ongoing tracking.

Detector Status Page The *Detector Status* page (as in Figure 4) visualizes the detailed information of the Detector node, one for each node. So, if there were five computers in the camera network there would be five pages.

The Detector Status page outline the processes for that node:

- if the ground plane is detected and the method being used (manual, assisted, automatic);
- the detection algorithm (HOG based, Munaro[11], YOLO);
- the frequency (Hz) of the RGB and Depth cameras;
- if people tracking and object tracking are running and at which frequency;
- a graph with the computer load and a graph of the number of detections;
- a log form with messages that explain the status of the node.

2.3 Integration with OpenPTrack

The integration of significant features in an existing software often required considerable changes also in the latter.

The user interface proposed is developed to be completely independent from the OpenPTrack system. This solution preserves the compatibility of the system, and, above all, it doesn't require any change to the current system.

The entire interface, indeed, works side by side with OpenPTrack, it collects data both from the camera network and the operating system and share these data using WebSockets protocol.

In order to clarify how the proposed interface integrates with OpenPTrack, an example is depicted in Figure 5. In this example the camera network is composed by 4 detectors and a tracker. All the computers are connected in the same network and share a common ROS Master to exchange the OpenPTrack data.

[11] A people detection algorithm developed by Matteo Munaro et al.[7]

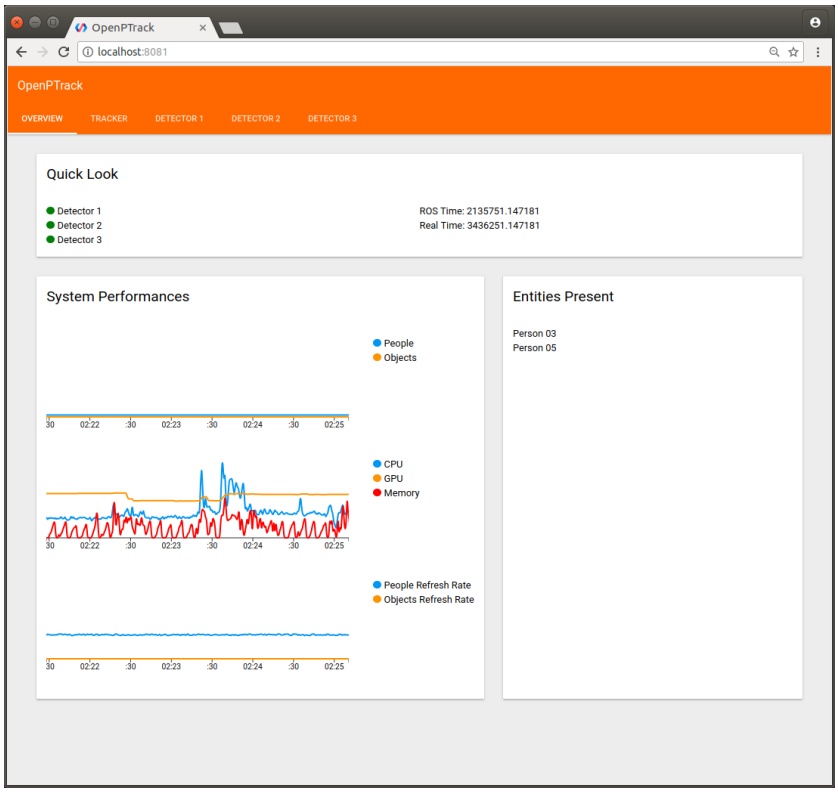

Fig. 3. The Overview page. On top, the web interface visualizes the status of each node, and the time sync between the operating system and ROS. The graphs show the number of people and objects tracked, the system load (CPU, GPU and Memory usage) and detection/tracking refresh rate. On the right, a list of the people and the objects tracked it is visualized.

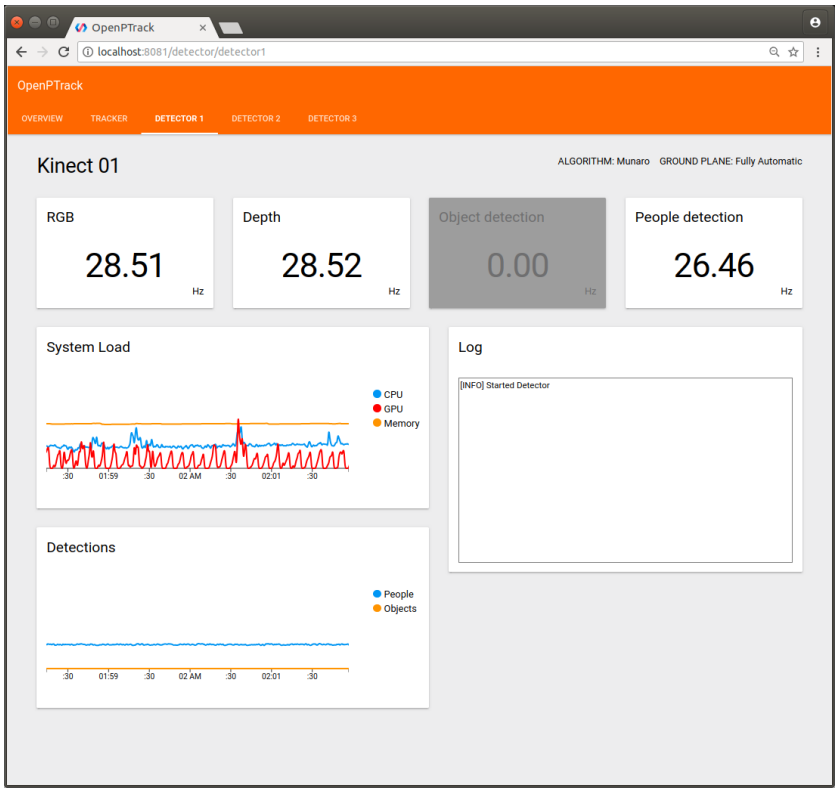

Fig. 4. The Detector Status page. On top, the web interface visualizes the camera publish rate and people and objects tracking frequencies if enabled (in this case objects detection are disabled). The graph on the top shows the system load (CPU, GPU and Memory usage) and the one on the bottom shows the detection frequencies. On the right a simple log form visualizes the most important messages from the system (in particular the emphasis is on error messages).

The web application runs in the same computer of the tracker, but it could also run in a distinct computer connected to the same network.

As explained in 2.1, in each detector and in the tracker run a NodeJS node that collects its own data and make them available through a WebSocket.

The web application is connected to each node thought WebSocket. Indeed, its configuration includes the IP addresses of each node in order to connect, ports instead are fixed (8080 for detectors and 8090 for trackers). Once the web interface opens a connection with a node, it receives data in asynchronous mode. In this way, only new data are exchanged between a node and the wed interface, thus avoiding overloading the network. Moreover, it is also possible configure the update frequency for each node and for each type of information.

WebSockets are bidirectional so in future by the user interface the user will be able to send commands to the OpenPTrack network.

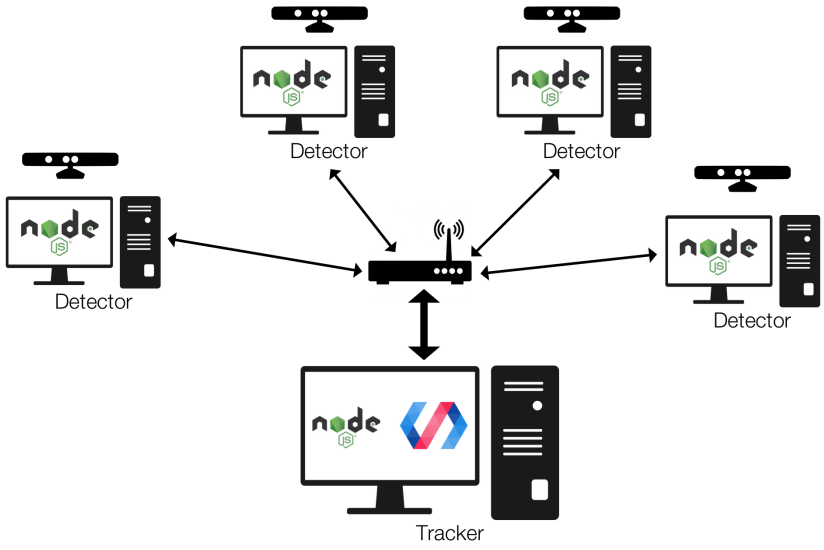

Fig. 5. An example of OpenPTrack camera network composed by 4 detectors and a tracker. All the computers are connected in the same network and share a common ROS Master located in the Tracker. In each detector and in the tracker run a NodeJS node that collects its own data of and makes them available through a WebSocket and the Polymer web application runs inside the Tracker.

3 Conclusions

This work presents a user-friendly user interface for OpenPTrack, a system for real-time people and object detection and tracking. It gives a clear view of the

status of an OpenPTrack network and allows to quickly recognize a fault and its cause.

This web interface is developed to be completely independent from the OpenPTrack system and it works side by side with the latter, collecting data both from the camera network and the underlying operating system and share these data using WebSocket protocol.

In order to be platform independent and modular, the presentation layer is developed using the Polymer library. Moreover, to maintain the scalability of the proposed system, each node (both detector and tracker) provide information of its own status thought a NodeJS node.

The proposed interface is actually in test both at the IAS-Lab of the University of Padova and at UCLA REMAP and will be publicly released as soon as a stable version is reached.

In the next future, we plan to integrate a visualizer that shows tracking in real time as the one in Figure 1. Moreover, additional debug details, such as more information on the tracker, will be added. A standard will be also defined for messages in order to help developers to expose new information or functions that actually are not shared between the OpenPTrack systems and the web interface.

Beyond the monitoring, in the future we aim for managing the entire OpenPTrack network through this web interface, starting from the integration of some maintenance tools such as a smart and simple calibration interface.

Acknowledgments

Authors want to thanks Randy Illum for numerous tests on the user interface.

References

1. Alexander, B., Hsiao, K., Jenkins, C., Suay, B., Toris, R.: Robot web tools [ros topics]. IEEE Robotics & Automation Magazine 19(4), 20–23 (2012)
2. Bradski, G.: The opencv library (2000). Dr. Dobbs Journal of Software Tools (2000)
3. Carraro, M., Munaro, M., Burke, J., Menegatti, E.: Real-time marker-less multi-person 3d pose estimation in rgb-depth camera networks. arXiv preprint arXiv:1710.06235 (2017)
4. Crick, C., Jay, G., Osentoski, S., Pitzer, B., Jenkins, O.C.: Rosbridge: Ros for non-ros users. In: Robotics Research, pp. 493–504. Springer (2017)
5. Munaro, M., Basso, F., Menegatti, E.: Openptrack: Open source multi-camera calibration and people tracking for rgb-d camera networks. Robotics and Autonomous Systems 75, 525–538 (2016)
6. Munaro, M., Horn, A., Illum, R., Burke, J., Rusu, R.B.: Openptrack: People tracking for heterogeneous networks of color-depth cameras. In: IAS-13 Workshop Proceedings: 1st Intl. Workshop on 3D Robot Perception with Point Cloud Library. pp. 235–247 (2014)
7. Munaro, M., Menegatti, E.: Fast rgb-d people tracking for service robots. Autonomous Robots 37(3), 227–242 (2014)

8. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA workshop on open source software. vol. 3, p. 5. Kobe, Japan (2009)
9. Rusu, R.B., Cousins, S.: 3d is here: Point cloud library (pcl). In: Robotics and automation (ICRA), 2011 IEEE International Conference on. pp. 1–4. IEEE (2011)
10. Toris, R., Kammerl, J., Lu, D.V., Lee, J., Jenkins, O.C., Osentoski, S., Wills, M., Chernova, S.: Robot web tools: Efficient messaging for cloud robotics. In: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. pp. 4530–4537. IEEE (2015)
11. Zhao, Y., Carraro, M., Munaro, M., Menegatti, E.: Robust multiple object tracking in rgb-d camera networks. In: Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on. pp. 6625–6632. IEEE (2017)