Proceedings of the 2014 IEEE
International Conference on Robotics and Biomimetics
December 5-10, 2014, Bali, Indonesia

Path Planning as a Service PPaaS: Cloud-based Robotic Path Planning

Miu-Ling Lam, *Member, IEEE* and Kit-Yung Lam

***Abstract*— Cloud computing framework can provide promising solutions for service robots to overcome various challenges in unstructured environments. In this paper, we present the design and implementation details of a novel cloud-based system architecture for robotic path planning. The proposed system, called *Path Planning as a Service* (PPaaS), is based on a three-layered architectural design: cloud server layer, cloud engine layer, and client robot layer. The cloud server layer is implemented with Hadoop Distributed File System. It contains a path plan database, which stores the solution paths that can be shared among robots. The cloud engine layer, which is the core of the architecture, utilizes Rapyuta cloud engine to create configurable containers for different robotic platforms. It also provides an on-demand path planning software in the cloud, which computes the optimal paths for robots to reach the goal positions. A load balancer is suggested to manage the container allocation in machines and coordinate local/cloud computation load balancing. The client robot layer is formed by a set of physical robots that employ PPaaS in the cloud. The communication framework of the entire system is based on Robot Operating System (ROS) platform, despite that robot clients do not necessarily run ROS. We have extended the proposed system for a case study of people tracking and following by multiple robots. We have also experimentally verified the feasibility and effectiveness of solving the shortest path problem via parallel processing in the cloud. The technical implementation details as well as the challenges of the proposed cloud robotics system are presented in the paper.**

I. Introduction

Over the past decades, industrial robots and manipulators have made substantial contributions for us. However, these robots are usually preprogramed and deployed only in structured environments. There is a strong demand for service robots that can assist human to perform various tasks intelligently and reliably. The major challenge for service robots is the ability to effectively interact and cope with the complexity and dynamics in human-inhabited or unstructured environments such as household, office and hospital. This involves computationally and data-intensive tasks, like navigation and object recognition, which require the robots to equip with very powerful onboard computing and memory resources.

On the contrary, there is an upward trend in the appearance of DIY robots. Several successful open-source embedded system projects such as Arduino, Raspberry Pi, Beaglebone, etc. have not only induced the evolutionary ‘maker’ movement in recent years, but have also made robot development become easier and more accessible for non-specialists. These extremely low-cost robots do not only bring pleasure for robotic hobbyists, but are also highly useful for researchers in performing experiments that require a large number of robots to study the topics of, for instance, swarm robotics and formation control of multi-robot systems. These robots are usually small, lightweight, and have minimal hardware configurations and, thus, limited processing power that is insufficient for handling compute-intensive tasks. The function range of these robots can be extended only if they can offload or outsource the difficult calculations and focus on sensing and actuations.

Cloud robotics is a promising solution for the above problems. It is an emerging paradigm extended from remote-brained robots [1], networked robotics [2], web-enabled robots [3], and the modern cloud computing framework. The cloud refers to the Internet and its associated datasets, services and users. Cloud computing technologies demonstrate a number of attractive features over traditional computing, such as resources sharing and cost reduction. Under the cloud computing framework, software, platform and infrastructure are delivered to users via the Internet as services. The continuous growth of ubiquitous networks and the demands for massive datasets and parallel computing resources for robots have evoked researchers to conceive that cloud robotics will play an important role in future robotic research. We believe that cloud robotics has much potential to accelerate the development of comprehensive software and large-scale databases that are useful for robots.

In recent years, a number of interesting projects in the domain of cloud-based robotics have appeared. The DAvinCi project [4] has proposed a cloud computing software architecture for service robots and has implemented a parallelizable simultaneous localization and mapping FastSLAM algorithm and messaging framework to demonstrate its effective performance gains. Other software architectures proposed for cloud robotics initiatives include [5], [6], and [7]. The RoboEarth project [8] and its cloud engine Rapyuta [9] provide a World Wide Web-style platform for robots to generate, share and reuse knowledge and data, and to access to robotic cloud services. Recently, the team has used some low-cost robots with depth cameras to perform collaborative 3D mapping in the cloud with Rapyuta [10]. Berkeley’s cloud-based robot grasping project [11] has exploited the Google object recognition engine to generate a set of candidate grasps stored in the Google cloud storage. The team has also demonstrated how parallel computation in the cloud can obtain optimal force closure grips in the presence of shape uncertainty through a parallelizable algorithm [12].

This paper is centered around the development of a cloud-based system, called *Path Planning as a Service*

The authors are with the School of Creative Media, City University of Hong Kong, Kowloon Tong, Kowloon, Hong Kong (e-mail: miu.lam@cityu.edu.hk).

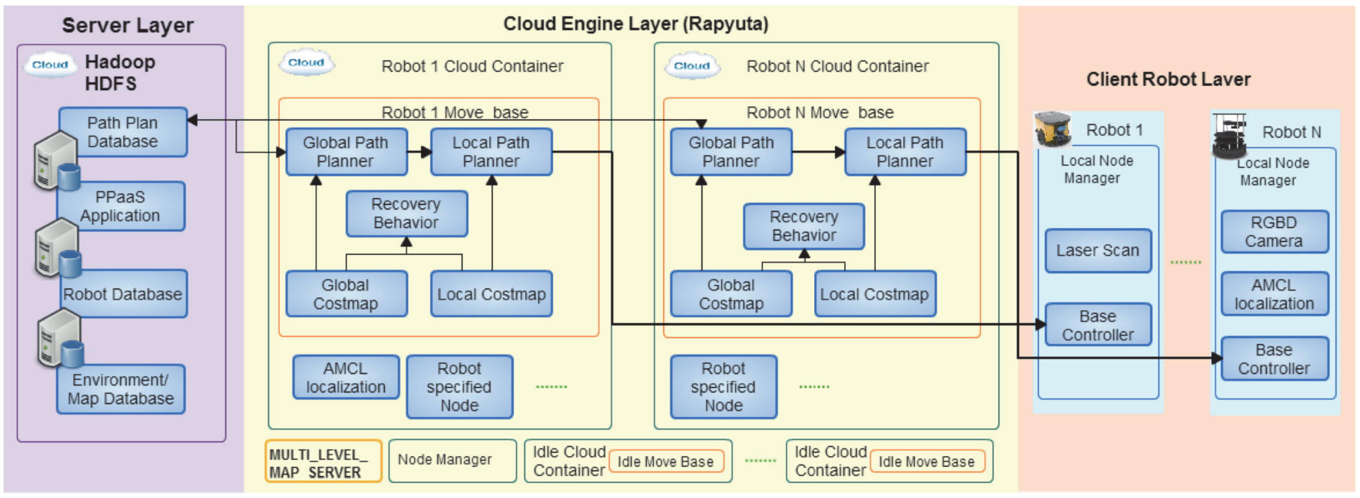

Figure 1. System Architecture of PPaaS.

(PPaaS), for robotic path planning. Path planning is an important robotic problem with many applications. In this work we focus on addressing the problem of computing the minimum cost navigation plan for mobile robot from a start point to a goal point in a map. This application can be easily extended for solving other related robotic problems such as motion planning of redundant manipulators. We propose a three-layered system architecture, which facilitates on-demand path planning software in the cloud. We have implemented the proposed system with the Rapyuta cloud engine and used Robot Operating System (ROS) [13] platform as the communication framework for the entire system.

The remainder of this paper is organized as follows: In Section II we give the overview of the system architecture with detailed description of the components in each layer. In Section III we present the implementation of multithread parallel processing of path planning algorithm. We also experimentally compare the runtime of single-thread local and multithread cloud-based computation with the implementation of the parallelizable Dijkstra's algorithm [14] [15] for solving the shortest path problem. In Section IV an enhanced load balancer in Rapyuta for cloud resources optimization and managing the trade-offs between local and cloud computations is presented. Section V describes the data storing mechanism of Path Plan Database. Then, in Section VI we provide a case study of utilizing the proposed cloud-based architecture in performing multi-robot people tracking and following. Finally we conclude in Section VIII with a brief discussion of future works.

II. System Architecture

The system architecture of the proposed cloud-based path planning system is illustrated in Fig. 1. It is a three-layered architecture, which includes a Server Layer, a Cloud Engine Layer and a Client Robot Layer. The detailed explanation of each component is as follows:

A. *Server Layer*

An important feature of cloud robotics is effective data and knowledge sharing among robots. The purpose of the Server Layer is to facilitate structured and massive datasets to be shared among robots. The proposed system uses Hadoop Distributed File System (HDFS) [16] for data storage among machines.

Robot Database stores the registered robot model, robot ID, robot owner ID, robot descriptions and Unified Robot Description Format (URDF) data, which is in XML format and used for representing a robot model. As the data model of each robot entry in the Robot Database is variable, flexibility of Cassandra makes a clear and convenient management.

Path Plan Database stores the computed path plans generated by the path planner of each robot. The data of each entry include the path length, start point, end point, and all waypoints in the path. Waypoints are stored as ROS geometry_msgs/Pose format. For convenient management, the portion of path plan waypoints that is duplicated with another path plan entry will be removed and replaced by the ID of that entry. The path data can be shared among the robots in the same map. More detail will be given in Section V.

Environment/Map Database stores the map data, which are used for navigation and AMCL (adaptive Monte Carlo localization). In our experiments, the map data are 2d image file in PGM format, which is readable by SDL_Image software, and are used by ROS map server.

B. *Cloud Engine Layer*

Rapyuta is used as the cloud engine platform for our PPaaS system. Each robot connected to PPaaS will have a dedicated computing environment implemented with Linux Containers (LXC) created for it. It provides a lightweight virtual machine, called *container*, to execute authorized programs. The machines can be added or removed conveniently via Rapyuta during runtime, thus providing a scalable grid-based computing model on-demand.

Node Manager contains a node monitor that works as a local/cloud load balancer. It determines whether a requested node is more suitable to be executed in the cloud engine layer or locally in the client robot's computing unit. The details of the load balancer will be discussed in Section IV.

Move Base is a component to perform cloud computation for robotic path and action planning. It is designed as an on-demand service (similar to the Software as a Service concept in cloud computing) in the cloud engine layer. Each robot has a dedicated Move Base that links together the Global Path Planner and Local Path Planner as soon as the robot cloud container is successfully constructed. Global Path Planner in Move Base computes the navigation plan requested by other ROS nodes either in the cloud or robot. Local Path Planner calculates and publishes the local path plan that tries to follow the global path plan. It computes velocity commands and sends them to the client robot's Base Controller. Base Controller subscribes to and executes the velocity commands, and publishes the feedback of command execution.

The Global and Local Path Planner are based on reuse-oriented design for component-based robotic systems [17]. Users can change and create path planner plugins according to their usage. New global path planner plugins should adhere to the base global planner interface and new local path planner plugins should adhere to the base local planner interface in ROS nav_core package. In our PPaaS system, each robot can choose the dedicated global path planner and local path planner without affecting other robots, by simply setting the value of the corresponding parameters to their preferred path planner plugin name, and some path planner specified parameters, either before or after the initialization of the robot cloud container. Move Base supports either 2D or 3D navigation. The 3D case can be used for complex environments with 3D collision checks.

A multi-layered Map Server is extended from the ROS map server and octomap server, providing interface for both 3D and 2D map data transferred from map database to the node. Fig. 2 shows an example of 2D occupancy grid map generated by ROS Gmapping and stored in the Map Server.

C. Client Robot Layer

Client Robot Layer is formed by a set of physical robots that employ PPaaS in the cloud. As PPaaS is based on Rapyuta, communication between physical robots and the cloud engine is established via WebSocket. Rapyuta's robot endpoint in the cloud uses JSON encoded messages. The client robots should be ROS-compatible in order to request for path planning service from PPaaS. They should be configured with registered user ID and password, connect to the PPaaS Rapyuta master and establish the WebSocket connection between robots and Rapyuta. By providing the necessary configuration data in JSON format, such as the registered robot model, robot ID, map ID, robot initial position in the map, a robot clone will be established in the cloud. The robot clone processes a set of procedures for each robot that are connected to the cloud, including managing key frames and other data accumulation tasks, and updating the robot with optimized (or post-processed) maps. The robot clone also sends the motion commands to the robot. By providing the goal position, the base controller of the robot should be able to execute the movement commands given by the cloud navigation stack.

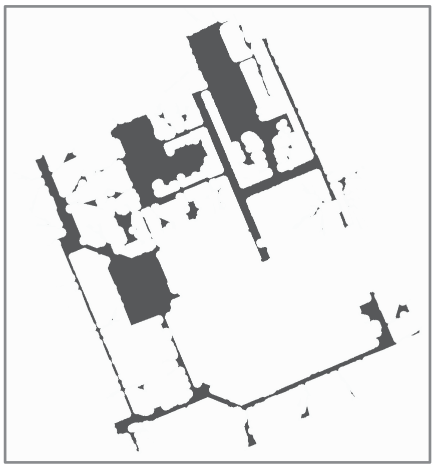
Figure 2. An example 2D occupancy grid map generated by ROS Gmapping.

III. Parallelization

Most of the CPUs today have multiple cores, which support multithread processing. In order to have a better utilization of computational resources, both Global and Local Path Planner are implemented with multithreading. Any inaccurate configuration of thread number, for instance, in case of thread number greater than the number of CPU cores, may lead to performance degradation of a multithread application. Therefore, each container in our system will be assigned with a limited size of CPU resources according to the request. The associated environmental variables of each newly constructed container will be set to the number of CPU core assigned. This can effectively constrain the maximum number of threads that a program can create. Beside the 2D navigation and 2D mapping, 3D navigation and 3D mapping, such as using the octomap 3D mapping library in ROS, are executable with multithreading in PPaaS.

We have experimentally compared the performance of single-thread local and multithread cloud-based computation with the implementation of the parallelizable Dijkstra's algorithm for solving the shortest path problem. The runtime of path calculations with ROS Navfn package in both computation models are recorded. In the multithread model, the functions for setting up navigation potential arrays, main propagation, and path construction are implemented with multithreading. To better illustrate the improvement purely caused by parallel processing and eliminate any artifact, all tests and the Rapyuta cloud engine are running in the same computer. The processor is an Intel Core Xeon E5-2630 v2 2.6GHz with twelve threads. The performance of parallelization is summarized in Fig. 3.

IV. Load Balancer

Rapyuta provides a simple load balancer for managing container construction and deconstruction in machines [9]. Each machine will set an upper limit to the number of containers that can be run simultaneously on the machine. A container with a bigger size requirement can get more computation and memory resources from the machine than a smaller size container does. The load balancer will assign new containers to the first machine in the list of registered machines if the available size in that machine is greater than the size required by the container.

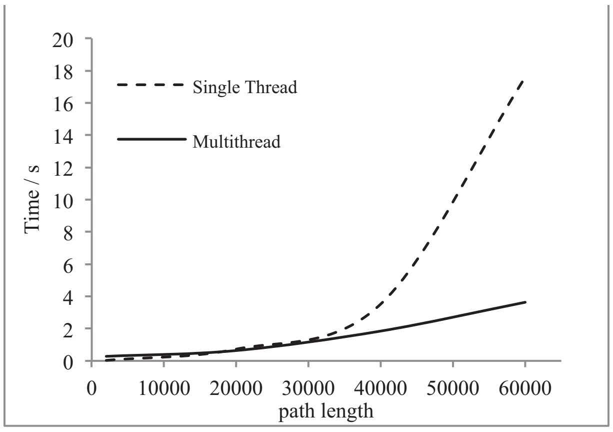

Figure 3. Comparison of single and multi-thread processing of shortest path problem in varying path lengths.

PPaaS has an enhanced version of load balancer. It includes the maps in use and the container group, and compares the utilization score and the network performance of all machines. Robots that use the same map need to get the same set of map data from the multi-layered Map Server. In such case, the Local Path Planner of each robot has to publish its position and velocity information and, at the same time, subscribe to the same information from other robots. Thus, the inter-robot communication is supposedly frequent.

We have performed extensive tests to measure the data communication round-trip times (RTT) occur in different scenario. According to our experiments, RTT between two nodes running in the same container is the smallest; the RTT between containers hosted in the same machine is smaller than that between containers hosted in different machines. Based on these findings, the PPaaS load balancer prioritizes the assignation of new robot containers to registered machines as follows:

If multiple machines have registered, the PPaaS load balancer will record the RTT test result. Containers for robots using the same map will be assigned to the same machine to minimize communication cost. If the quota of machine is full, new container will be assigned to the machine with the smallest utilization score, which is an index to find out the best-fit machine with the least utilization rate based on the average RTT, average network bandwidth, average CPU and memory resources utilization rate of the machine.

Additionally, the enhanced load balancer has a node monitor to suggest whether the user specified node should be run in the cloud or in the robot's local computing unit, provided that the robot client computer is able to run that node. If the node is suggested to be executed locally, the node monitor will inform the local node manager. If the execution of node on robot fails, node monitor in the cloud will arrange the node to be executed in a specified robot container.

Three criteria for suggesting whether a node should be run in the cloud or on robot are summarized as follows:

1. Local computation power
   The computational capacity of robot constrains the number and size of programs that it can execute. For instance, a low-cost, single-board computer with single core CPU, like Raspberry Pi, may not be able to run Move Base and OpenNI [18] tracker simultaneously. Node monitor will give such robot the highest priority to execute the node in the cloud. The priority list is based on the computational power of local computers and it is customizable.
2. Node properties
   Historical average RTT of data communication between robot and cloud container for the same node running in the cloud is longer than that running on robot. However, node requiring high computational resources should have higher priority to be executed in the cloud. In addition, nodes that are time-critical for communication with robot should be executed locally.
3. Network bandwidth
   Another important consideration is the bandwidth of the network between robots and the PPaaS server. If a node subscribes to the message data that occupies a large portion of the bandwidth for transmission, which is published by robot, the node should be executed on robot. Taking image processing node as an example, it requires the whole frame of raw image for computation. Thus, the node should be computed in robot local device such as the OpenNI camera with raw image compressor. Table I shows the bandwidth usage of OpenNI camera image in form of ROS message image data. For 2D navigation stack used by many ROS-compatible robots, AMCL is used to provide probabilistic localization of the robot. It subscribes to the laser scan data provided by robot laser range finders or depth camera, where depth image is converted to laser scan data by depthimage_to_laserscan node. The published scan data is about 80 KB/s in general. Robots equipped with depth camera and using AMCL localization in the PPaaS system are suggested to execute the depthimage_to_laserscan node locally in order to reduce the bandwidth occupied by transferring the raw depth image.

TABLE I. BANDWIDTH OF IMAGE STREAM AT 30FPS

	Depth image (size depends on complexity of scene)	*RGB image*
Raw	18.5-20 MB	~10 MB
Compressd	1-2 MB	2-2.5 MB

V. PATH PLAN DATABASE

Path Plan Processor provides the interface for create, read, update and delete (CRUD) operations for Global Path Planner or any node requesting service from the path plan database. It searches the matching path plan in database with the same start and end points in the request. If no matching path is returned, the Global Path Planner will compute the requested path plan and store it as new path plan entry in database. Then, it finds the existing path plans with duplicated waypoints, and replaces the duplicated portion of new path plan with the IDs of the existing paths, as show in Fig. 4.

Every path plan entry will be examined and partitioned in a constant granularity level G if duplication is found. G is also the minimum length of any new entry. The maximum number of duplicated partitions in a new entry is its path length divided by G. If the length of duplicated waypoints between two entries is bigger than G, the duplicated portion will be extracted as a new entry.

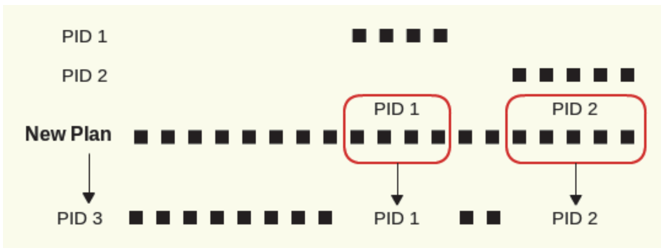

Figure 4. Path plan database data storage mechanism.

Path plan data will be used for replacing the planning process of Global Path Planner when all of the three criteria below are satisfied:

1. The data transmission time is not longer than the estimated computation time of the planner for calculating the path with the same complexity.
2. Current map version used for navigation is the same as the path plan data in the database since the map with small modification may result in a different path.
3. The algorithm used for generating the path stored in the database is the same as the algorithm used in current path planner, unless it is proved to be a better algorithm than current algorithm, which provides a shorter path with the same start and end points.

With the reference of historical path plan data generated for different robots, the path planning algorithms of the Global Path Planner can be evaluated.

VI. Case Study: Multi-Robot People Tracking and Following

In this section, we extend the proposed system architecture for performing multi-robot people tracking and following as an application example of PPaaS (Fig. 5). An extra node, People Following Coordinator, is running in the cloud engine layer for making the decision of which robot in the team to follow a human target. Human target can be defined as either the person at the nearest position to one of the robots in the team, or a specific person exists in the scene and has been pre-defined by the user for the system. In our experiment, we have adopted the former case. The People Following Coordinator analyses and concludes all detected human torsos, then assigns which robot to follow the target based on different rules. The rule used in our case study is that the robot with the minimum cost (shortest path to target) will be assigned to follow the target.

For the client robot layer, two Turtlebot robots [19] R_1 and R_2 are used in the experiment (Fig. 6). The functionality for human target tracking using the Kinect depth camera is supported by OpenNI framework and NiTE with camera stabilization [20]. Each robot will publish the coordinate frame(s) of the person (people) detected in the map via ROS tf (transform) package with frame id as {Robot ID}/{Torso ID}. Base controller subscribes to the action command and publishes the feedback of command execution.

Global Path Planner in Move Base computes the navigation plan for following the human target assigned by the Coordinator. After the computation of navigation plan, Local Path Planner makes and publishes the real-time action command for the robot.

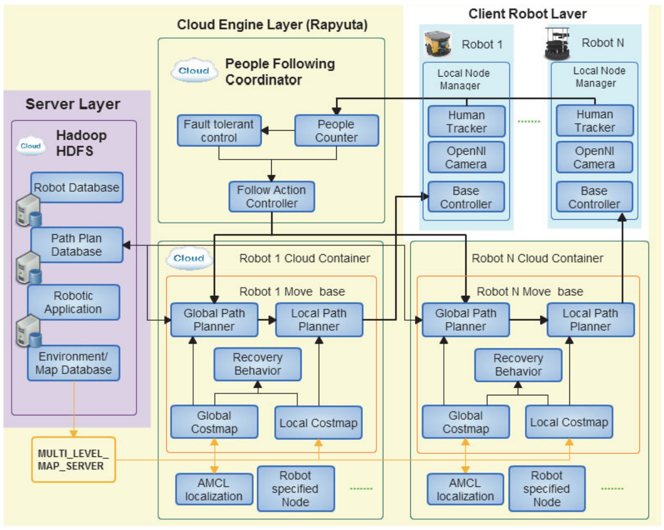

Figure 5. System architecture for multi-robot people tracking and following

The procedures of People Following Coordinator for assigning robot to follow target are:

1. Find the maximum position deviation of each robot in the map.
2. Subscribe to torso coordinates from all robots that have detected human. Transform all torso coordinates from robot coordinate frames to the world coordinate frame.
3. Recognize any identical person detected by different robots in the map and count the true, total number of people identified by all robots (People Counter).
4. Follow Action Controller compares the shortest path from each robot to each person identified by that robot. The paths are generated by Global Path Planner using the Dijkstra's algorithm. Then, assign robot R with the shortest path to follow the human target.
5. Observe the historical cost of each robot for target following; switch to robot R' only if the cost of R' is strictly decreasing and is smaller than R.

VII. Conclusion

In this paper, we presented a cloud robotic system called Path Plan as a Service (PPaaS). The system adopts the cloud computing framework for providing path planning service for ROS-compatible robots. The three-layered system architecture is presented with detailed descriptions of major components. Based on the RTT measurements, we have listed the realistic considerations and trade-offs between local and cloud computation for the sake of load balancing. We have implemented multithread processing for path construction and verified the effectiveness of the approach in solving the shortest path problem for a large map in the cloud. We have also successfully extended PPaaS for a case study of people tracking and following by multiple robots. In future work we will improve the implementation of parallel computing using more powerful platforms and include other parallelizable path planning algorithms. We are also interested to extend the function range of the proposed system to cover more robotic applications.

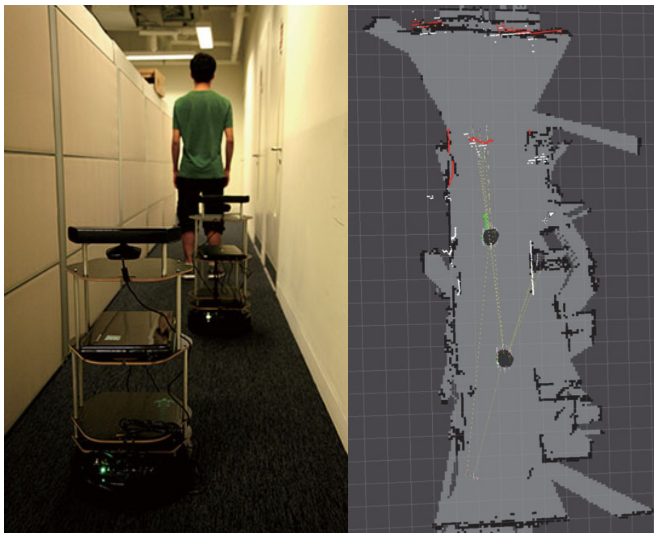
Figure 6. Multi-robot people tracking and following.

ACKNOWLEDGMENT

This work was supported in part by the Croucher Foundation under Grant 9500016 and City University of Hong Kong under Grant 7004160.

REFERENCES

[1] M. Inaba, "Remote-brained robotics: Interfacing ai with real world behaviors," *Robotics Research,* vol. 6, pp. 335-344, 1994.

[2] A. Sanfeliu, N. Hagita, and A. Saffiotti, "Network robot systems," *Robotics and Autonomous Systems,* vol. 56, pp. 793-797, 2008.

[3] M. Tenorth, U. Klank, D. Pangercic, and M. Beetz, "Web-enabled robots," *Robotics & Automation Magazine, IEEE,* vol. 18, pp. 58-68, 2011.

[4] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong*, et al.*, "Davinci: A cloud computing framework for service robots," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010, pp. 3084-3089.

[5] Z. Du, W. Yang, Y. Chen, X. Sun, X. Wang, and C. Xu, "Design of a robot cloud center," in *Autonomous Decentralized Systems (ISADS), 2011 10th International Symposium on*, 2011, pp. 269-275.

[6] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: Architecture, challenges and applications," *Network, IEEE,* vol. 26, pp. 21-28, 2012.

[7] K. Kamei, S. Nishio, N. Hagita, and M. Sato, "Cloud networked robotics," *Network, IEEE,* vol. 26, pp. 28-34, 2012.

[8] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez*, et al.*, "Roboearth – a world wide web for robots," *Robotics & Automation Magazine, IEEE,* vol. 18, pp. 69-82, 2011.

[9] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea, "Rapyuta: The roboearth cloud engine," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA), Karlsruhe, Germany*, 2013.

[10] Cloud-based collaborative 3d mapping with rapyuta using low cost robots. Available: https://github.com/rapyuta/rapyuta-mapping/wiki/

[11] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg, "Cloud-based robot grasping with the google object recognition engine," in *IEEE Int'l Conf. on Robotics and Automation*, 2013.

[12] B. Kehoe, D. Berenson, and K. Goldberg, "Toward cloud-based grasping with uncertainty in shape: Estimating lower bounds on achieving force closure with zero-slip push grasps," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012, pp. 576-583.

[13] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs*, et al.*, "Ros: An open-source robot operating system," in *ICRA workshop on open source software*, 2009.

[14] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik,* vol. 1, pp. 269-271, 1959.

[15] A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders, "A parallelization of dijkstra's shortest path algorithm," in *Mathematical foundations of computer science 1998*, ed: Springer, 1998, pp. 722-731.

[16] Apache hadoop. Available: http://hadoop.apache.org/

[17] D. Brugali, L. Gherardi, A. Biziak, A. Luzzana, and A. Zakharov, "A reuse-oriented development process for component-based robotic systems," in *Simulation, modeling, and programming for autonomous robots*, ed: Springer, 2012, pp. 361-374.

[18] Openni framework. Available: http://www.openni.org/

[19] Turtlebot open-source robot development kit. Available: http://www.turtlebot.com/

[20] T. Naseer, J. Sturm, and D. Cremers, "Followme: Person following and gesture recognition with a quadrocopter," in *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS'13)*, 2013.