

TritonBot: First Lessons Learned from Deployment of A Long-term Autonomy Tour Guide Robot

Shengye Wang Henrik I. Christensen*

Abstract—Service robots perform flawless demos while their developers are keeping a close eye, but they often fail when working autonomously, especially in a long-term deployment. To study failure modes and human-robot interaction patterns in long-term deployments, we built TritonBot, a long-term autonomy robot working as a building receptionist and a tour guide. It recognizes people’s face, talks to them, and guides people to the labs and facilities in an office building. This paper presents the design of TritonBot and the lessons we learned from the first-month deployment with respect to technical and human-robot interaction aspects. TritonBot and its variant BoxBot have worked for 108.7 hours, actively interacted with people for 22.1 hours, greeted people 2950 times, guided 150 tours, and traveled 9.9 kilometers. We share the components of TritonBot using an open licence to help the community to replicate the TritonBot platform and inspire long-term autonomy and human-robot interaction research.

I. INTRODUCTION

Robot technology has advanced significantly over recent years, and we have finally reached a milestone where we can apply these innovations in real-world applications. However, while robots often perform flawless demos and win applause under controlled settings with operator’s guidance, they frequently fail when working for an extensive length of time in unstructured environments without close supervision [1]. Long-term deployment adds more uncertainties and covers many corner cases in the technical components or human-robot interaction behaviors. As a result, long-term autonomy is a significant challenge that prevents service robots from entering an office or home environment to assist people.

We present TritonBot, a long-term autonomy robot deployed in an unsupervised open environment. TritonBot works as a receptionist and a tour guide of a university office building. It recognizes people’s face, talks to people, and shows people the labs and facilities in the building. However, TritonBot is not only another tour guide robot: In addition to serving the visitors to the building, the systems is used to discover failure modes in a long-term deployment of a robot that actively interact with people in an open environment. Besides, we also plan to use TritonBot as a toolbox to discover the short-term and long-term interaction patterns between the robot and humans since the robot receptionist engages in many interactions every day. Fig. 1 shows TritonBot and its variant, BoxBot. Despite the difference in the shapes, BoxBot runs the identical software and exhibits the same behavior as TritonBot.

* Department of Computer Science and Engineering, University of California, San Diego. La Jolla, CA 92122, USA. {shengye, hichristensen}@ucsd.edu



Fig. 1. TritonBot (right) and its variant, BoxBot (left). TritonBot is based on Fetch Research Edition [2] and BoxBot is a Freight robot with an upper body made of cardboard boxes. They take turns and work every weekday, chat with people and navigate with people to introduce place of interest in an office building.

In the first month deployment, TritonBot and BoxBot received much attention from visitors: They have been serving the guests for 108.7 hours in total, actively talked and walked with people for 22.1 hours, provided 150 tours, traveled 9901.1 meters, have been listening for 4.0 hours (2,616 counts) and speaking for 10.7 hours (14,037 counts). Unlike other long-term autonomy robots, TritonBot is a standalone service robot system that actively engages people and draws people’s attention. It interprets users’ intent only using voice interaction and talks back to people. With the deployment, we were able to study potential issues in resilience, scalability, and learning of a robot system in a dynamic environment. We released the components in TritonBot under an open source licence to enable the community to replicate our platform for long-term autonomy and human-robot interaction (HRI) research.

This paper is structured as follows: Section II discusses some existing long-term autonomy robots and other related work. Section III presents TritonBot’s behavior, components, and its ability to use as a toolbox for HRI studies. Section IV discusses the lessons we learned in both technical and human-robot interaction aspects of TritonBot from the first month of deployment. Section V presents a quantitative analysis and statistics on TritonBot’s performance, and Section VI concludes the paper.

II. RELATED WORK

TritonBot shares its goals with a few pioneering long-term autonomous systems. Valerie [3], the “Roboceptionist”

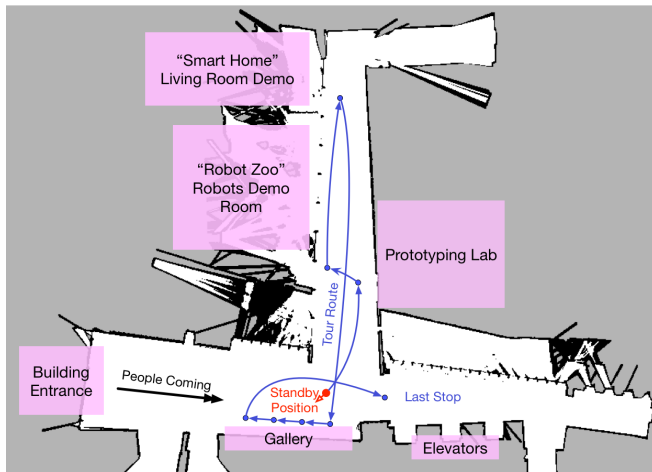


Fig. 2. 2D map of the hallway in the office building. The robot keeps a standby pose when it is waiting for people. The blue trajectory represents the route that the robot guides people, and the blue vertex indicates the places of interest where the robot will stop and introduce.

at the Carnegie Mellon University is an early robot with a personality that exhibits social competence and remains compelling to interact with for an extended period. CoBots [4] is a long-term autonomy system that intensively study long-term mapping, localization, and navigation in an unstructured environment; they have traveled over 1,000 km in total under unstructured environment. The European STRANDS project [5] deployed four robots for security and elderly care scenarios and reached a few weeks of autonomy within a single deployment. BWIBots [6] is a custom-designed multi-robot platform for AI, robotics, and HRI that aims to be an always-on, permanent fixture in a university building.

Early tour guide robots like RHINO [7] and Minerva [8] provided interactive tours to visitors in museums with an emphasis on robust localization and navigation in a crowded environment. The SPENCER project [9] investigates social interaction in a crowded environment with multiple humans, which is critical in an unsupervised open setting. Kanda et al. deployed Robovie robots in a corridor of a shopping mall for five weeks to provide information to the public [10].

Interactive robots have appeared in many other scenarios. Komatsubara et al. use the Robovie robots in an elementary school for five weeks with the intent to increase the curiosity in the science of children [11]. The robot recognizes the face and asks questions to the students, but it requires operator's intervention for speech recognition. Bohus et al. created a stationary directions robot to study open-world human-robot interaction [12], and they concluded several failure modes [13]. Chung et al. studied the Savioke Relay robots in hotels and created a fast-prototyping tool to program the robots [14]. Tonkin et al. studied social robots at airports and discussed design methodology for designing communication pattern for interactive robots [15].

TritonBot shares a few similarities with these pioneers, but also has its specialty. TritonBot only interacts with a human through speech, therefore it receives many unexpected instructions, and they help the researchers to understand

people's free expectation on a robot. Also, TritonBot has long-term memory, recognizes people, and interacts with people every day, therefore it can collect and analyze communication pattern and histories, which makes it a toolbox for future system and HRI studies.

III. SYSTEM DESIGN

The TritonBot operates in the hallway of an office building, discovers passersby with face detection and interacts with them using speech. This section describes the scenario of TritonBot deployment and the robot's behavior, components, and its capabilities as a research toolbox.

A. The Tour Guide Scenario

TritonBot works as a receptionist and a tour guide of Atkinson Hall, a six-story, 250,000 sq ft building at UC San Diego. The building is an office building for regular users, but sometimes hosts meetings, conferences, or events. Fig. 2 shows a map and the surroundings of the area where the robot is deployed. TritonBot stands in the hallway of the building facing the entrance at an angle that sets the camera away from facing the entrance where excessive outdoor sunlight is coming from. TritonBot continuously detects human faces in its view, and when it sees a person, it actively engages the person by greeting him or her, but it gives up if the person disappears before it finishes a self-introduction monologue.

If TritonBot finds a match of a recently seen face with a record in its face database, it will greet the person by name. Otherwise, it starts asking for name and offers a few trivia questions. Fig. 3 shows an example of the interaction between TritonBot and a visitor. The quizzes are simple, but they provide a time frame when the visitor is looking at the robot. TritonBot continuously collects face samples while talking to the person and uses the highest-quality face samples to update its face database. Afterwards, TritonBot offers a guided tour to the user. If the user accepts the tour, the robot will show the visitor around and introduce a few places of interest in the building, including a fast-prototyping facility with 3D printers, a robot showroom (robot zoo), a smart home demo room, and a gallery (Fig. 2). If the robot fails to move to a target, it politely asks people to step back because it "needs personal space to maneuver."

We try to keep TritonBot autonomous as much as possible to study failure cases. TritonBot is programmed to go to a charging dock when the battery is low, and it returns to work when it finishes charging. However, in the early deployment, we closely monitor the robot to prevent any accidents: A student always sits in the "smart home" room when TritonBot is actively working. He monitors the robot and manually overrides the robot control in case of emergency. Although the student never engaged the emergency stop in the first month, the working time of the robot was limited to the student's schedule.

B. Components in TritonBot

The initial TritonBot and the variant, BoxBot are based on Fetch Research Edition and Freight robots from Fetch

[TritonBot]: Good morning. My name is TritonBot. I am a robot working here at the Contextual Robotics Institute. May I ask what is your name?
 [Human]: My name is April.
 [TritonBot]: Nice to meet you, April. I prepared some quiz for you. Would you like to give them a try?
 [Human]: Sure.
 [TritonBot]: Where does the president of the United States of America live?
 [Human]: The White House.
 [TritonBot]: You got it! The president of the United States of America lives at the White House in Washington DC. Do you want to try another question?
 [Human]: Yeah.
 [TritonBot]: What is the most popular sport throughout the world?
 [Human]: I don't know.
 [TritonBot]: No worries. Soccer is the most popular sport throughout the world. Do you want to try another question?
 [Human]: No, thanks. I'm good.
 [TritonBot]: Thanks for chatting with me. I feel honored. I am a tour guide. Can I show you around?
 [Human]: Yes, please.
 [TritonBot]: Glad to hear that. Please follow me.
 (TritonBot moves towards the Prototyping Lab.)
 [TritonBot]: We are now in the Prototyping Lab. The prototyping lab provides prototyping and engineering services to UCSD researchers, including embedded electronics, mechanical devices, robotic contraptions, real-time networking software, and 3D printed structures.
 (TritonBot moves towards the Robotic Zoo.)
 [TritonBot]: Here is the robot zoo. You can see my fellow robot, Diego-san. He can see people, understand gestures and expressions, and even learn from people. Diego-san is not the only robot here, many of my other fellow robots are coming very soon.
 (TritonBot introduces a few other locations and stops at the elevators.)
 [TritonBot]: Here is the elevator, and it is our last stop. Thanks for letting me show you around. Please enjoy your stay. Take care and have a great day.

Fig. 3. A dialogue between TritonBot and a visitor. TritonBot starts the dialogue when it sees a face, but gives up if the face disappears during its self-introduction. When TritonBot matches a face with an acquaintance, it will greet the person with the name and skip the trivia questions.

Robotics Inc. respectively [2]. BoxBot has a custom upper body made of cardboard boxes. The cardboard box upper body allows researchers to change the design or adding new sensors to the platform with a minimal cost. These two robots take turns in the deployment, which opens time windows for hardware maintenance and software upgrade.

TritonBot has onboard embedded computers with x64 processor, and it utilizes many third-party components and open-source software to accomplish its task. It is running Robotic Operating System (ROS) Indigo version (released in 2014) [16]. Openface [17] pipeline detects human faces in the pictures captured by the camera and converts these face images to embeddings that are used to calculate the similarity between two faces. Cartographer [18] provides localization and mapping for the TritonBot platform, and the ROS navigation stack provides lower-level control for the robot to move around. Cloud service comes with excellent help when an open-source alternative is not available or does not meet our needs. Google Cloud Speech API [19] translates speech audio to text with nearly real-time feedback. TritonBot used an Android tablet for voice synthesis initially, but we switched to a commercial solution later because of its superior performance.

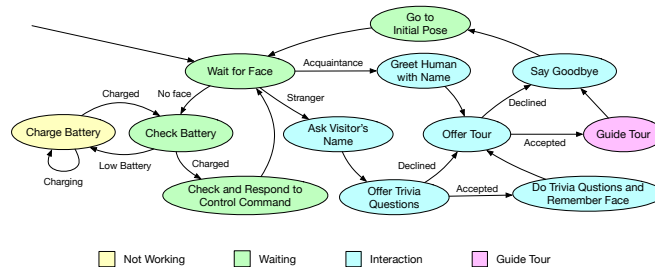


Fig. 4. The state machine that represents the behavior of TritonBot. Different color indicates different state categories. The time spent in each category is shown in Fig. 8.

C. TritonBot for Human-Robot Interaction Studies

TritonBot works closely with people, and the long-term deployment allows it to meet people and interact with them over a long timespan. Although the early deployment of TritonBot is targeting long-term autonomy research only, we believe TritonBot is capable of distinguishing people and studying the human-robot interaction behaviors.

Working as a robot receptionist, TritonBot starts interacting with a person when it sees a face, which could provide an estimation of the age, gender, or other background information of a person [20]. Since TritonBot works in an office building, it can see a relatively fixed set of people. In a long-term deployment, the robot can observe the change in the way that the same person interacting with the robot. The robot can also observe a general group of people and study their interaction with the robot.

The tour guide scenario is a test field for social-aware navigation research. TritonBot observes the environment using a laser scanner, a microphone, and a camera, which provides an estimation of the number of the humans around and their pose [9]. While a user is following TritonBot during a tour, the robot observes the relative location between itself, the person, nearby obstacles, and incoming people, which allow the robot to learn the rules in social-navigation.

Although people's curiosity about the robots dominates the interaction pattern in the early deployment, we expect to see a noticeable difference in the interaction pattern of TritonBot and BoxBot in the future and improve the robot shape and design.

TritonBot is designed as a portable and replicable HRI research platform, and we open-source TritonBot to provide a research platform for HRI and long-term autonomy researchers. Most components we built for TritonBot are open-sourced either as ROS packages or standalone components and are available at <https://github.com/CogRob/TritonBot>.

IV. LESSONS LEARNT FROM EARLY DEPLOYMENT OF TRITONBOT

In a month-long deployment of TritonBot, we have seen people enjoy chatting and walking with a robot, but we have also seen many shortcomings in the system. These shortcomings include failures and faults, improper or inefficient robot operation workflows, and imperfections in

human-robot interaction behaviors. This section presents the main lessons we learned from the first-month deployment of TritonBot.

A. Hardware Failures

Just like any other engineered system, failure is unavoidable in robotics. In contrast to software errors, hardware failures are usually uncommon, but the long-term deployment extensively tests the system and reveals failure cases. During the development and deployment of TritonBot, we encountered a few hardware issues:

1) *Battery failure:* During our early deployment, the BoxBot over-discharged its batteries (two 12V lead-acid batteries), and the failed batteries could only support the robot for 10 minutes (as opposed to 12+ hours of standby time in normal condition). We replaced the faulty batteries and set up a battery monitoring program to prevent battery from over-discharging.

2) *Electrical part failure:* The charging port on TritonBot failed, and the robot attempted many times going back and forth trying to dock itself to a charging dock when the battery was low. The student monitoring the robot discovered the unusual behavior and shutdown the robot program. After that, we programmed the robot to shutdown itself when its battery is low and auto-charging fails.

3) *Mechanical failure:* The connection between a caster and the base on TritonBot became loose during long-term but normal operation. We fixed the issue and notified the manufacturer.

4) *Device driver failure:* The robot computer fails to recognize a USB camera sometimes during the deployment, and reconnect the USB connector did not solve the issue. The only known working solution was to reboot the internal computer, and we ended up replacing the camera with another model.

Since hardware errors are usually hard to recover from automatically, fail-safe is a critical design principle: the robot should detect failures and take actions to prevent failure propagation or further damage. Component-level monitoring is helpful sometimes: A battery voltage monitor will prevent the batteries from over-discharge, and a retry limit on the docking action and a vibration sensor will prevent further physical damage to the robot.

Because hardware malfunction is less usual, they are easy to ignore and not covered by most test cases. In fact, the TritonBot control software was programmed to give up and report docking failure after five docking attempts, but the higher-level control software immediately restarted docking because the battery-level was still low. The hardware failures lead us to the need for extensive testing, which requires a fault injection/simulation mechanism to produce/reproduce potential errors.

B. Network Connectivity

TritonBot accesses the Internet via a wireless connection to reach cloud services, receive commands from operators, and report status to the monitoring station. Since it roams

in a large area of the building and a campus-wide wireless network already exists, setting up an ad hoc wireless network for it is not worthwhile. Using an existing wireless infrastructure with multiple access points and broad coverage leads us to a situation that (1) signal strength is not consistent in all the locations, (2) the robot sometimes loses connection when it roams between multiple access points, (3) network is not secured.

We took a few measures to confront these issues. First, we did a signal strength survey in the building and excluded blind spots from the tour path. Besides, the robot checks network connection every minute and resets its wireless interface when there is a failure. We also installed a wireless access point on the robot to provide “escape hatch” access to the robot in case of unrecoverable connection failures. Furthermore, we set up a firewall on the robot to block unexpected incoming connections from the Internet.

C. Software Failures

In a developing system, software errors are frequent when rolling out new features, improvements, or bug fixes. While most of the software flaws disappear in the testing stage, the remainders are usually hard to discover or even to reproduce. However, they are easy to recover from by restarting the program, and the loosely-coupled system architecture in ROS enables such recovery mechanism without causing a system-wide outage. Nevertheless, this mechanism requires coordination between the software components. From our experience, we have found two design principles are particularly useful to tolerate and recover from rare and transient software errors:

- 1) Any component must tolerate temporary unavailability of other components and should not propagate the failure. They should resume working if the depending component recovers.
- 2) Restarting a program shall always help it to enter a clean and steady state. If a component fails in some way and cannot recover, it is better to terminate and restart the program than to remain in an unknown and unresponsive state.

These two principles allow the system to tolerate transient errors. In a typical scenario, the erroneous programs terminate and return to a normal state; meanwhile, the counterparts will wait until they recover rather than propagating the error. Moreover, these design rules decouple the programs at the application level, which allows programs to regularly restart when the system is idle, as Google points out that “a slowly crash looping task is usually preferable to a task that hasn’t been restarted at all [21].” As a bonus, these design principles also enables dynamic updating of the system at the cost of temporary and short unavailability.

These two principles helped the robot running stable for most of the time. But a slight violation of the first principle caused unavailability of our system that it could not automatically recover from: One day, the behavioral control component tried to communicate with a logging service that timed out, and the behavioral control stalled.

The same symptom happened three times during the entire month deployment, and we discovered the error message in the last occurrence and fixed the control state machine.

This incident leads us to another lesson: we assumed the reliability of some software components (the logging service in this case) since they rarely fail. We are developing a mechanism to inject errors into software components, which will help discover potential design flaws.

D. Software Deployment

TritonBot's ROS system is running about 50 nodes when it is working, and more than half of them are in-house customized software. Initially, we ran these programs directly under a Linux system, but this method does not scale. As we were adding more components to the system, software dependencies started to conflict with each other, runtime configurations scattered at multiple locations in the file system, and software versions were hard to track. After testing several solutions, we ended up using Linux containers (Docker [22] specifically) to manage the robotic software in TritonBot.

Linux containers provide operating-system-level virtualization and allow building and shipping applications easily as self-contained archives. Pioneer work introduced Linux containers technology to robotics [23], but to the best of our knowledge, TritonBot is the first robotic system that uses Linux containers to manage the entire robotic software stack. Every TritonBot robotic software component is running inside a container, including the ROS master node and the robot drivers. These containers share the host networking stack to allow smooth communication. A container orchestration tool, `docker-compose` helps managing runtime parameters. All the ROS runtime configurations, including the `roslaunch` files, are collected in a single version-control system, which allows the developers to trace the complete change history of configuration change through the robot's life cycle. We arrange the container images in a hierarchy so that some images share the same base image with common libraries, and updating a base image will also update its dependents.

In short, Linux containers help to manage robotic software on a developing and scaling system, help to maintain high availability, limit fault zones, and allow easy replication and distribution of robotic software.

E. Navigation

TritonBot uses Cartographer [18] for localization and map creation and uses the ROS navigation stack, "move base" to move around. On top of the navigation stack, we created a topological map to guide the robot to predefined routes, as shown in Fig. 5. The topological map contains waypoints and path: a waypoint represents a featured location such as an intersection of aisles or a place of interest, and a path between two waypoints is line-of-sight that the robot can easily travel. A navigation planning service determines a series of waypoints for the robot to go through before it

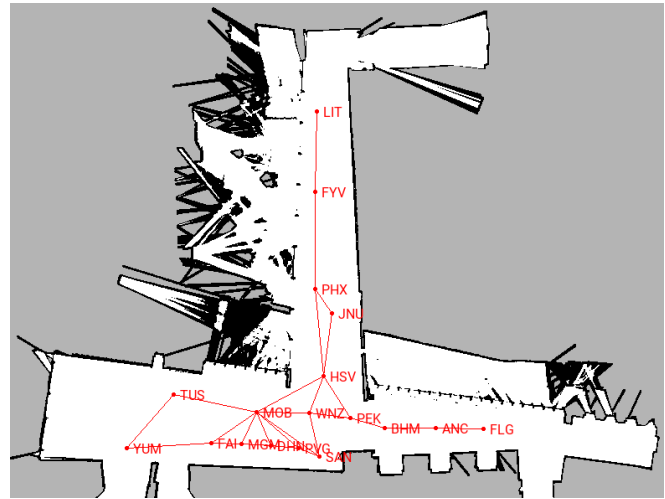


Fig. 5. The topological map that TritonBot uses to navigate. Waypoints are marked with three-letter names, and paths connect the waypoints that the robot can move between.

reaches a destination. Each path has an estimated travel time and is updated when a new estimate is available.

The two major failure modes in TritonBot's navigation behavior are localization failure and execution failure. Most 2D SLAM software matches current laser scan with a pre-built map to estimate the current location of the robot, and when the robot is surrounded by many temporary obstacles (e.g., humans), it may match the observation with an incorrect location. We only observed this failure once when the robot was surrounded by about ten people, and the robot quickly corrected the issue afterward as it moved around a bit and gained more observations. A sudden change in the robot's location estimation is usually an indicator of a localization failure. On the other hand, dynamic obstacles, especially humans are the biggest challenge in robotic navigation. Some users tried to block the way of the robot to test the robustness of the robot. When TritonBot fails to move, it says "I need some personal space to maneuver, can you please step back." In the most case, people will yield to the robot, and the robot resumes after executing some recovery behavior.

Another lesson we learned from robot guiding person is that the person sometimes leaves the robot in the middle of a tour. As a result, the robot continues to introduce the place of interests to nobody. Without full user study it is hard to conclude why the users leave the robot, but this situation hints that the robot should respond to the person leaving or joining the tour. As an improvement, we are integrating leg detection into the system and use it to confirm the person is still around during the navigation.

TritonBot moved 9901.1 meters in the first month of deployment. Among the 1,722 attempts moving from a location to another, 97.4% attempts were successful, and 44 attempts failed. Most of them are because of timeout (32 attempts) and move failures (8 attempts) due to dynamic obstacles partially or fully blocking the robot's path. Other failures are because of cancellation command (three attempts, expected behavior) or temporary unavailability in topological

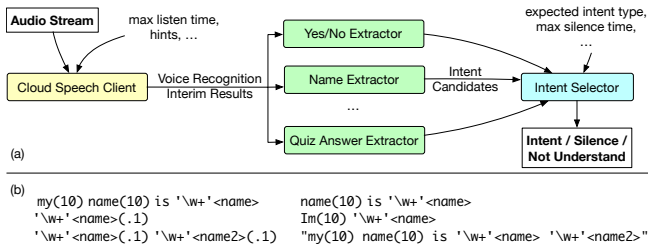


Fig. 6. (a): Speech recognition pipeline for TritonBot. Google Cloud Speech client converts audio stream to transcripts, and intent extractors interpret the transcript and generate intent candidates with confidence scores. The intent selector chooses an intent candidate based on expected intent type, time, and other criteria. The intent selector also stops listening if it can return a result. (b): Templates used in “name extractor.” Name extractor is a template-based intent extractor that converts the response to “what is your name?” to structural intent representation. Each word in the template is `regex<capture name>(weight)`, and the template is matched against voice recognition result using longest common subsequence algorithm to calculate a confidence score based on `weight`. Words with `capture name` can be used to fill in some fields of an intent.

map service (one attempt). Moving from a location to another requires traversal of several paths in the topological map (Fig. 5), and TritonBot and BoxBot traversed 4,876 paths. Traveling path PHX-FYV and FYV-LIT caused 20 failures in total. We did not collect enough data to conclude an exact reason, but we did see some people trying to block the path of the robot in this spacious aisle, and only walked away when the robot requested space to maneuver.

F. Speech and Dialogue

TritonBot has a voice recognition pipeline that converts audio stream to user intent data structure as shown in Fig. 6(a). TritonBot uses Google Cloud Speech API to convert voice input from the microphone [19], which returns real-time interim results and rich metadata about the speech, including a confidence score and start/end time for each of the words in the utterance. A few intent extractors convert utterance transcript to candidate intents. The intent selector either yield an intent data structure or special intents for “silence” and “not understand.” The intent selector also stops the robot’s listening when it can return an expected result so that the robot can respond quickly. Most of the current intent extractors are implemented with matching the words in the input sequence to regular expressions in a template sequence using longest common subsequence algorithm, and Fig. 6(b) shows an example.

An interesting observation is that sometimes people respond to questions, especially yes-no questions, even before the robot finishes talking. As a result, the robot misses the response because it was not able to capture the very first part of the utterance. In late deployment, we changed synchronization mechanism to allow the voice recognition to start 0.3 seconds before the robot finishes speaking according to previous studies [24], so that the robot can capture the full response from the human but almost none of the speech from itself.

G. Face Recognition

The TritonBot has a head camera (Primesense Carmine 1.09) at 1.5 meters from the ground facing 15 degrees

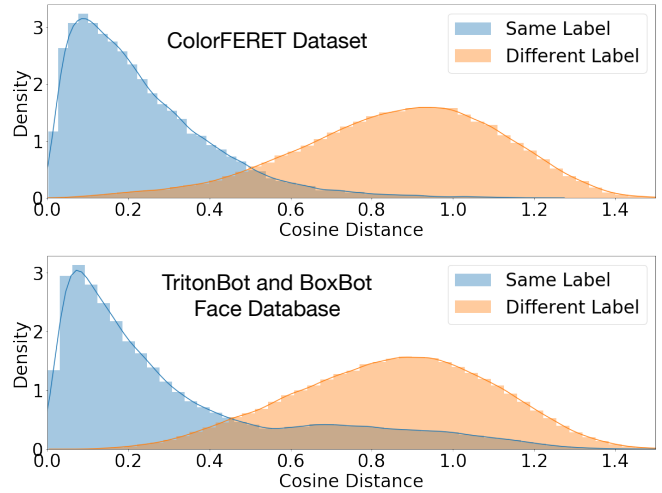


Fig. 7. The distribution of cosine distance of the OpenFace embeddings, between the faces of the same person and the faces of the different persons in the ColorFERET [25] dataset and TritonBot and BoxBot’s face database.

upwards, but the camera mounted on BoxBot is at 1.2 meters height facing about 30 degrees upwards. Both TritonBot and BoxBot can see an undistorted face, but BoxBot cannot detect people standing too close (less than 0.5 meters) from the robot because of the tilt of the camera.

TritonBot uses deep neural networks based on Openface [17] to recognize visitors’ face. As a starting point, we use the nearest neighbor to find the best match given a face. When the cosine distance ($d(\mathbf{A}, \mathbf{B}) = 1 - \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$) is between the detected face and the candidate is greater than a threshold, the face recognition reports an unknown face indicating a stranger. The threshold was initially set to 0.5 by balancing false positive and false negatives on the ColorFERET dataset [25], and we will adjust to reflect the actual dataset for the next deployment. Fig. 7 shows the distribution of cosine distance of the Openface embeddings of the same person and different persons from the ColorFERET dataset, and the actual faces TritonBot and BoxBot remember.

While TritonBot usually recalls the name of a seen person, as expected the mis-classification ratio increased as the database size is increased. One of the reasons is that sometimes people face the robot sideways, and the robot could only see a side face, which is easy to confuse on different people.

H. Logging

Logging in long-term autonomy robots provides opportunities for data analysis, testing new software, and replaying the scene. ROS provides a logging utility, `roscat` to log general communication during the robot operation. We create a program to record a substantial portion of ROS communication in TritonBot, including laser scanner data, localization estimation, the command to move the base, etc. Even though the robot only records `roscat` files when it is actively interacting with a person, we have collected 367.8 GB “roscat” files in the first month of deployment.

Although “roscat” records many details in ROS communication, it is far from ideal for recording internal states

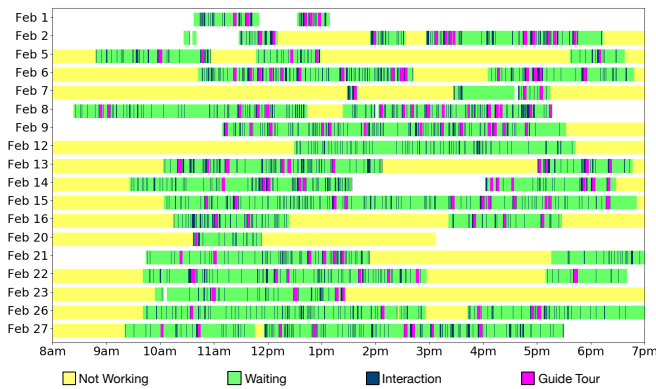


Fig. 8. Working log of the robots. Feb 1, Feb 8, and morning Feb 13 came from TritonBot, and the others came from BoxBot. Blank indicates the robot is under maintenance or taken to special events and the top-level control software is not running, and light yellow background means the robot is charging itself. The robots have worked for 108.7 hours (waiting, interaction, and guiding tour), actively interacted with people for 22.1 hours (interaction and guiding tour).

for software components in a long-term autonomy robot because of two major flaws: (1) Changing the log record fields definition in `rosbag` invalidates all previous logs, which is inconvenient for an actively-developed long-term autonomy robot. (2) `rosbag` is designed to capture the communication between ROS components, and it is costly to send massive internal states through network sockets when they only meant to be written to a disk.

To overcome these deficiencies, we created a logging library, “Universal Logger” to save internal states of both ROS and non-ROS components. Universal Logger records structural log data in an efficient and backward-compatible Protocol Buffer [26] format. While maintaining strong-typed structure, Protocol Buffer also allows adding new fields without affecting previously serialized logs. Every robotic component using Universal Logger writes logs to the disk independently without going through any communication interface. Also, Universal Logger automatically creates new binary log files when date changes or a single file is larger than a certain size, and we arrange these files in directories with the creation date. For example, speech recognition logs of Feb 27, 2018, on BoxBot is stored in the following path: `boxbot/2018/02/27/dialogue/speech_recognition/1519752112.pb.gz`. The logs are transferred to a storage server and removed from the robots everyday to save the storage space. So far we have collected 2.8 GB data in Universal Logger format in the first-month deployment, and use them to study long-term autonomy (and conclude the result in this paper).

I. Safety

Safety comes first when a robot is deployed in an open environment. Mobile platforms of TritonBot and BoxBot have a built-in hardware level safety mechanism that slows it down when the laser scanner detects movements nearby, but a grad student always monitors the robot physically close to the robot for the first-month deployment to avoid unexpected

malfunctions. However, in the first-month deployment, the guardian never engaged the emergency stop mechanism. There was one case that a child tried to put his foot under the robot, but the robot slowed down immediately, and the child stepped back by instinct.

Requiring in-place human supervision harms the purpose of long-term autonomy experiments, since the robot working time is limited to the guardian’s work time. With more confidence in robot’s safety, we will slowly move to remote monitoring: we have created a private website to show the vitals of the robot and the camera image for safety monitoring purpose, and we deployed remote e-stop buttons that engage run-stop mechanism on the robot via the control network. We are also setting up bump sensors on the robot to stop its motion when it hits something.

V. RESULTS

In the first-month initial deployment, TritonBot and BoxBot together worked 18 days in total, skipping holidays and special events in February 2018. TritonBot only operated two and a half days because of maintenance and other projects requiring the platform, but BoxBot worked for 15 and a half days. Fig. 8 shows the breakdown of the activity of the robots. The two robots have been serving the guests for 108.7 hours in total, actively talked and walked with people for 22.1 hours, and spent 8.25 hour in guiding people around.

TritonBot and BoxBot remembered 167 persons with 8,681 face samples, and they did 97,284 face detection and recognitions. As for speech, they had been listening for 4.0 hours (2,616 counts) and speaking for 10.7 hours (14,037 counts). The robots greeted people 2950 times. It recognized acquaintances 416 times, although the false positives are not determinable at the current times. In the 434 attempts of asking a visitors’ name, it heard a response 239 times. The robots played 133 trivia games and asked 307 trivia questions. Among the 292 answers, 224 were the correct answer, 47 were wrong answers, and the user indicated they did not know the answer in the other 21 responses.

The robot moves when they guide people to places of interest or need to go to a charging dock. The robots traveled 9.9 kilometers during the deployment, giving 150 tours in total. It attempted to move to a specific location (a waypoint in the topological map, see Fig. 5) 1,722 times, and 1,678 (97.4%) attempts were successful. To accomplish these targets, it traveled 4,876 edges in the topological map, spent 6.1 hours in total.

VI. CONCLUSION

Long-term deployment reliability is one of the most crucial parts of any feasible autonomous service robots. TritonBot served as a study in robot-human interaction patterns and long-term service viability before they can enter our homes and businesses. This paper discussed TritonBot’s design and summarized lessons we learned from the first-month deployment of TritonBot.

Just like any engineered system, failures are unavoidable in robotics. As a design principle, the robot design should

prevent hardware failures from propagating and cause further and/or permanent damage to the robot. Software failure should be tolerated in a loosely-coupled system since most of the failures can recover by restarting a program. More importantly, testing helps discover design flaws, and fault-injection/simulation in hardware, software, and networking will help to discover and test failures in robotics beforehand. System engineering helps to reduce the effort of maintaining a long-term autonomy robot development platform, and Linux container significantly decreased the stress of managing a scaling robot system.

The most significant challenge of robots is still human-beings. From our observation, people tried to block the path of the robot, try to fool the robot in talking, and ignores the robot talking to itself when they leave. Sometimes humans unintentionally face the robot side-wards, expect the robot to respond when it is working on something else, or talk to the robot in a way that it could not understand. Long-term deployment exposes all of these cases and helps us design better robot behavior.

Our future goal is to improve TritonBot to meet and exceed the expectations of its users. We aim to formalize research questions based on our first-month observations and continue the long-term deployment of TritonBot to study its autonomy for a longer timespan. After all, when a robot enters our homes and businesses, it should not require frequent maintenance like automobiles.

REFERENCES

- [1] H. I. Christensen, Ed., *A Roadmap for US Robotics: From Internet to Robotics*, Nov. 2016.
- [2] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch & freight: Standard platforms for service robot applications," in *Workshop on Autonomous Mobile Service Robots, International Joint Conference on Artificial Intelligence*, July 2016.
- [3] R. Kirby, J. Forlizzi, and R. Simmons, "Affective social robots," in *Robotics and Autonomous Systems*, vol. 58, Pittsburgh, PA, March 2010.
- [4] S. Rosenthal and M. M. Veloso, "Mixed-initiative long-term interactions with an all-day-companion robot." in *AAAI Fall Symposium: Dialog with Robots*, vol. 10, 2010, p. 05.
- [5] N. Hawes, C. Burbridge, F. Jovan, L. Kunze, B. Lacerda, L. Mudrova, J. Young, J. Wyatt, D. Hebesberger, T. Kortner, R. Ambrus, N. Bore, J. Folkesson, P. Jensfelt, L. Beyer, A. Hermans, B. Leibe, A. Aldoma, T. Faulhammer, M. Zillich, M. Vincze, E. Chinellato, M. Al-Omari, P. Duckworth, Y. Gatsoulis, D. C. Hogg, A. G. Cohn, C. Dondrup, J. P. Fentanes, T. Krajnik, J. M. Santos, T. Duckett, and M. Hanheide, "The strands project: Long-term autonomy in everyday environments," *IEEE Robotics Automation Magazine*, vol. 24, no. 3, pp. 146–156, Sept 2017.
- [6] P. Khandelwal, S. Zhang, J. Sinapov, M. Leonetti, J. Thomason, F. Yang, I. Gori, M. Svetlik, P. Khante, V. Lifschitz, J. K. Aggarwal, R. Mooney, and P. Stone, "Bwibots: A platform for bridging the gap between ai and humanrobot interaction research," *The International Journal of Robotics Research*, vol. 36, pp. 635–659, 02 2017.
- [7] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "The interactive museum tour-guide robot," in *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, ser. AAAI '98/IAAI '98. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1998, pp. 11–18.
- [8] S. Thrun, M. Benezewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz, "Minerva: a second-generation museum tour-guide robot," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 3, 1999, pp. 1999–2005 vol.3.
- [9] O. A. I. Ramrez, H. Khambhaita, R. Chatila, M. Chetouani, and R. Alami, "Robots learning how and where to approach people," in *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, Aug 2016, pp. 347–353.
- [10] T. Kanda, M. Shiomi, Z. Miyashita, H. Ishiguro, and N. Hagita, "A communication robot in a shopping mall," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 897–913, Oct 2010.
- [11] T. Komatsubara, M. Shiomi, T. Kanda, H. Ishiguro, and N. Hagita, "Can a social robot help children's understanding of science in classrooms?," in *Proceedings of the Second International Conference on Human-agent Interaction*, ser. HAI '14. New York, NY, USA: ACM, 2014, pp. 83–90.
- [12] D. Bohus, C. W. Saw, and E. Horvitz, "Directions robot: In-the-wild experiences and lessons learned," in *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, ser. AAMAS '14. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 637–644.
- [13] S. Andrist, D. Bohus, E. Kamar, and E. Horvitz, "What went wrong and why? diagnosing situated interaction failures in the wild," in *Social Robotics*, A. Kheddar, E. Yoshida, S. S. Ge, K. Suzuki, J.-J. Cabibihan, F. Eyssele, and H. He, Eds. Springer International Publishing, 2017, pp. 293–303.
- [14] M. J.-Y. Chung, J. Huang, L. Takayama, T. Lau, and M. Cakmak, "Iterative design of a system for programming socially interactive service robots," in *Social Robotics*, A. Agah, J.-J. Cabibihan, A. M. Howard, M. A. Salichs, and H. He, Eds. Springer International Publishing, 2016, pp. 919–929.
- [15] M. Tonkin, J. Vitale, S. Herse, M.-A. Williams, W. Judge, and X. Wang, "Design methodology for the ux of hri: A field study of a commercial social robot at an airport," in *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, ser. HRI '18. New York, NY, USA: ACM, 2018, pp. 407–415.
- [16] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.
- [17] B. Amos, B. Ludwiczuk, and M. Satyanarayanan, "Openface: A general-purpose face recognition library with mobile applications," CMU-CS-16-118, CMU School of Computer Science, Tech. Rep., 2016.
- [18] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1271–1278.
- [19] Google LLC. (2017, May) Cloud speech api – speech to text conversion powered by machine learning. [Online]. Available: <https://cloud.google.com/speech>
- [20] Y. Fu, Y. Xu, and T. S. Huang, "Estimating human age by manifold analysis of face pictures and regression on aging features," in *2007 IEEE International Conference on Multimedia and Expo*, July 2007, pp. 1383–1386.
- [21] B. Beyer, C. Jones, J. Petoff, and N. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, Incorporated, 2016.
- [22] Docker Inc. (2018, Nov.) Docker is an open platform to build, ship and run distributed applications anywhere. [Online]. Available: <https://www.docker.com>
- [23] R. White and H. Christensen, "Ros and docker," in *Robot Operating System (ROS): The Complete Reference (Volume 2)*, A. Koubaa, Ed. Springer International Publishing, 2017, pp. 285–307.
- [24] T. Stivers, N. J. Enfield, P. Brown, C. Englert, M. Hayashi, T. Heineemann, G. Hoymann, F. Rossano, J. P. de Ruiter, K.-E. Yoon, and S. C. Levinson, "Universals and cultural variation in turn-taking in conversation," *Proceedings of the National Academy of Sciences*, vol. 106, no. 26, pp. 10 587–10 592, 2009.
- [25] National Institute of Standards and Technology. color feret database. [Online]. Available: <https://www.nist.gov/itl/iad/image-group/color-feret-database>
- [26] Google Inc. (2018, Apr.) Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data. [Online]. Available: <https://developers.google.com/protocol-buffers/>