Abstract—This paper provides details on a low-cost 3D mapping robot called the TortoiseBot. Based on the Willow Garage TurtleBot platform, TortoiseBot uses off-the-shelf parts and is programmed on top of Willow Garage’s Robot Operating System (ROS). This design addresses shortcomings of the TurtleBot by offering the expandability to add additional sensors, computational power, or extended battery life. The robot is capable of creating dense point clouds of indoor environments and is an ideal low-cost platform for research on new localization, mapping, and planning algorithms as well as active sensor and collaborative robot research.

INTRODUCTION

Accurate and detailed three dimensional maps are very useful to fields such as surveying, defense, security, robotic rescue, and automation. Robots use these maps for navigation and localization in complex environments and it is a key component for full robot autonomy in unstructured environments [1]. People use maps for recording and representing scenes and locations. These maps are based on point-clouds, or datasets of Cartesian points. Color information can be combined with the map to make photo-realistic 3D models [2]. This eases recognition of objects and places in the map. Thus, the pursuit of three dimensional (3D) mapping and navigation of robots for indoor and outdoor environments is an active area of research [3], [4], [5], [6], [7], [8].

Until recently, devices for capturing 3D maps were expensive and required significant setup and human-control [9]. For example, tripod mounted scanners such as the Faro Focus [10] require being moved several times to make a complete map of an area. Individual scans are not combined together until later at a computer so the final map is topically not available until sometime after the scanning process. These units also cost forty to fifty thousand dollars. Other robotic platforms with mapping capabilities like the Google Street View trike and trolley [11], Segway RMP [4], ROAMS [2], AVENUE [12], PR2 [13], and Junior [14] provide very good solutions but are either too bulky to navigate through common indoor areas and/or use very expensive 3D scanning systems hardware such as SICK or Velodyne LIDAR sensors (Table I).

The topic of how to create and make use of these 3D maps is still one under heavy research. As such, it is beneficial to have an affordable, flexible, and reliable platform to run experiments on. Our goal then, is to create just such a platform to enable this research. The ideal robot should use off the shelf hardware with minimal customization and the plans should be available for anyone to freely use. All software should be open-source, which is free for anyone to modify and re-use. It should be mobile enough to drive around different terrains and output a map or model of the environment. Most of the data collection and processing should be autonomous. The design should be upgradable to match the mission requirements. This will help keep the base configuration of the robot affordable.

TORTOISEBOT

We call our solution to the ideal, low-cost mapping robot design problem the TortoiseBot, as shown in Figure 1. The robot is based on the TurtleBot design by Willow Garage [15]. Both robots use some of the same major off-the-shelf components such as the Microsoft Kinect and iRobot Create.
Create. The user community has developed a very thorough open-source Robotic Operating System (ROS) \[16\] software stack for the TurtleBot which we have modified to run on TortoiseBot.

**Hardware Components & Options**

The major components of TortoiseBot are the iRobot Create mobile robot platform, the vision sensors (Microsoft Kinect and Hokuyo LIDAR), and the onboard computer. They are listed in Table II and displayed in Figure 2. We opted to design our own computer system rather than use an available single-board-computer or laptop computer. There are benefits and drawbacks to this decision. The major benefit is the option to easily expand and adapt the computer with different components. This is rarely possible with laptops. We can choose components that are as fast as the application requires. If those components become outdated they can be easily changed. Each component is chosen with regard to price, performance, and energy consumption.

Our price goal was to be less than the cost of a laptop of comparable performance. The performance goal is to use \( \leq 25\% \) of processing power while running all the required hardware drivers and networking. That goal would ensure that the programmer has a healthy overhead to run their own programs within.

**Power Consumption:** Because the robot operates under limited battery power, the power consumption of every component must be taken into consideration. The CPU takes by far the most power but various potential 500mA USB loads can add up quickly. The computer has an extremely flexible power supply which can run from a 6-24 Volt source. This allows the user to supplement the robot’s power with any number of rechargeable battery or direct current options. For example, while the computer does not need to be mobile (while programming), it can be powered by a typical benchtop power supply found in many electronics laboratories or an appropriate AC to DC converter. The computer components are chosen to have high performance and relatively low power consumption. Under full load, with all peripherals operational, the computer draws 3 Amps at 12 Volts. This results in a battery life of one hour (Table III), which is plenty of time for running an experiment.

**Manufacturing and Hardware Modifications:** One of our goals was to keep the hardware modifications and fabrication to a minimum thus making the platform accessible to a wider audience. Only the three plastic structural plates need to be manufactured. They are cut out of sheets of ABS plastic using a 60 Watt laser cutter (Universal Lasers Systems Inc. VLS6.60). The bottom plate holds the computer in the cargo bay of the Create. The second plate supports the Kinect, and holds the cables above the computer. Plate two has a large vent above the CPU so that heat can escape. The topmost plate has a grid of holes .75 inches apart. This allows sensors to be easily attached to the robot.

The Kinect and the Create both require some slight modification to work on the robot. The power cable of the Microsoft Kinect needs to be modified to accept a 12 Volt input from the computer power supply. This can be easily done with a pair of wire strippers. As previously mentioned, the computer requires 3 Amps of current. Unfortunately, the power supply in the cargo bay of the Create can only safely supply 1.5 Amps. To solve this problem, the Create must
be disassembled and leads must be soldered directly onto the battery terminals so that the computer can receive full power from the battery. This operation is also easy because the Create is only held together with 16 screws and the area to be soldered is clearly labeled “Battery In.”

**Performance Benchmarking:** We compare our computer system against a common dual-core netbook computer which is included with Turtlebot, and a full-size laptop (Dell XPS 15Z) which closely matches the performance of our custom solution. We compare raw processing power using results submitted by users of PassMark’s synthetic CPU benchmark (http://www.passmark.com/about/index.htm). The results show the two more powerful systems to be almost five times quicker than the Turtlebot netbook. This makes sense as they are newer and use more modern CPU features like automatic overclocking (turbo-boost) and 32nm lithography.

**Comparison to Existing Systems:** The benchmark and feature comparison show that the Dell XPS 15Z and Tortoisebot custom computer have similar performance capabilities but our robot has no screen, self-contained battery, keyboard, or trackpad. This helps reduce the computer footprint and reduces costs to slightly more than half that of the laptop. A computer like this is called “headless,” that is, it does not have any visual display or keyboard. This does not affect normal operation of the robot because it displays results and get all input from a remote computer. The custom computer’s largest part is the Mini-ITX motherboard which is still smaller than any netbook. As a result, it fits in the cargo bay of the Create, a space which is used for bundling extra cables in the original Turtlebot design. This lowers the robots center of gravity and makes it more stable.

**Other Hardware:** The mobile robot base is an iRobot Create robot [17] with a laser-cut ABS plastic structure to hold the computer and sensors in place (Figure 2). The vision sensors are the most expensive part of the robot. The Microsoft Kinect[18] is used for color imaging and medium accuracy depth measurement. The high-accuracy vision sensor is a fixed position LIDAR (Light Detection and Ranging) sensor. Depending on the required range and outdoor or indoor environment, the appropriate LIDAR sensor can be selected from those available in Table I. The faster the scan speed, the more scans the robot will be making while driving and the denser the final point cloud will be. The TortoiseBot uses the Hokuyo URG-04LX-UG01 laser scanner. It was chosen because the Create is restricted to moving on smooth indoor surfaces so the laser scanner needs only to have indoor capabilities and range. The 100 msec scan time corresponds to a 10 Hz scan rate. This is acceptable for the speed the Create moves at. The TortoiseBot platform can also support communication hardware, that will enable us to build wireless networks among multiple robots, for autonomous coordinated sensing tasks.

**SOFTWARE**

The onboard computer runs ROS (Robot Operating System)[16]. It is an open-source robotics solution that provides libraries and tools to help create robot applications including hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. Because it’s free to use and adapt, many drivers and algorithms for common tasks are available for immediate use and customization.

ROS is able to use a model of the robot’s joints and parts to track individual movements and the resulting coordinate frame transformations. The model is an XML format file called URDF (Unified Robot Description Format) as seen in Figure 3. The model is generated of the robot using real-world dimensions and simplified geometry. Besides providing a nice visual representation of the robot, the model accurately provides the position of each part relative to the part it is attached to.

Driver processes run to process the data coming from the robots sensors. The data is tagged with its source in the URDF model so that the appropriate coordinate frames
can be transformed by the tf (transform frame) library. This allows data to be interpreted in the correct frame. For example, the lasers scans could be projected either vertically or horizontally relative to the robot depending on how the sensor is mounted.

A process called Robot Pose EKF (Extended Kalman Filter) combines wheel odometry, gyroscope data, and visual odometry using an Extended Kalman Filter to calculate the full six dimensional (x,y,z, roll, yaw and pitch angles) pose of the robot. Because the sensors are not perfect and there are extreme conditions in the real world, there is imprecision and noise in all data. This combination of different data sources provides the best position estimation possible. For outdoor applications, GPS information can also be input and fused for more accurate position estimation. Once the above drivers are configured and running, low level functions of the robot, like turning individual wheels, are abstracted away and the programmer can simply tell the robot where it should go. This is very similar to the type of abstraction that an operating system provides on a computer, allowing the programmer to use convenient structures like arrays and data structures instead of flipping bits and allocating memory.

APPLICATION TESTING

To test the usefulness of our platform, we ran realistic workloads that robots must process to map and interact with their environment. We chose to run the mapping and navigation programs described below.

Driver Performance

The absolute baseline of computer performance for the robot is to run the operating system, hardware drivers, and network communication. These are all required for any higher level programs to run. We ran this baseline workload for 15 minutes and recorded the average system usage. The computer has two cores so it has a maximum CPU usage of 200%. Our test ran with an average load of $20/200 = 10\% \leq 25\%$ which is our goal value.

Two Dimensional SLAM

A ROS process called SLAM (Simultaneous Localization and Mapping) reads odometry data and laser scans to localize the robot and generate a two dimensional map of the environment [19]. The SLAM algorithm takes individual horizontal lasers scans and uses them to build a map of the environment. Later the robot can match scans against this map to localize itself on the map with high precision. This algorithm can be run either using the Kinect sensor or by mounting the laser scanner horizontally. The computer is able to build maps and localize with a load of 45%. Figure 4 describes the features of navigating with this system executed on TortoiseBot.

Three Dimensional Point Cloud Generation

Adding another dimension to the 2D SLAM algorithm means either using a 3D sensor like the Kinect or sweeping a 2D sensor like the laser scanner. We chose to sweep the vertical laser scanner to generate a three dimensional point cloud as the scanner pans and rotates with the robot. The laser scans at a rate of 10 Hz. First, the scans are filtered. There are inaccurate outlier points in each scan which should be removed. Also, points in the scan that are parts of the robot (self-scanning) should be removed. Each scan is tagged with the timestamp and combined with the orientation of the robot at that time. This allows the scan to be transformed by tf from a set of ranges and angles into a set of three dimensional Cartesian coordinates called a point cloud. Now that a point cloud has been obtained, multiple clouds can be combined into one large and dense scan. An image of one of these clouds is shown in Fig. 5. The final point cloud map can be saved as a rosbag or a PCD (Point Cloud Database) file to be manipulated by software like Autocad or Pointools.
To test the accuracy of this method of point cloud generation, we scanned several rooms and household objects. After the point cloud was build, we compared dimensions of the physical object with the measured dimension. Results from that comparison can be seen in Table IV. We experimented with different size targets, scanning the target multiple times, and rotating at different speeds. These tests proved that the system can handle these complex tasks. The recorded point clouds consisted of up to 100,000 points. Thus, TortoiseBot is able to process large data sets.

With a camera correctly aligned with the laser scanner, it is possible to embed the corresponding RGB value into every point so that the point cloud may display a full-color model of the space. The addition of color information makes it much easier for a person to recognize parts of the cloud as specific objects. This data may also improve future computer vision algorithms such as object recognition. For example, a color difference in a kitchen point cloud might help a robot distinguish between a cabinet and a refrigerator. TortoiseBot is expected to be able to easily handle this type of task.

The robot is currently operated by driving the robot using a joystick or keyboard. When a scanning location is found, the robot is commanded to spin in place with specified speed and direction. The speed varies the density of the resulting point cloud. A sample map built out of several scans is shown in Figure 6. It should be noted that the distance between points in every individual scan is dependent on the angular resolution of the laser scanner. Most scanners vary between 0.25° and 0.36° of accuracy. The horizontal resolution of the scan is a function of the speed the LIDAR is panned across the scene. It is possible to have lasers which scan four times faster and thus would have four times as many points at the same panning speed. It is also an option to sweep the laser against the target area multiple times to fill in more and more fine details. The only potential problem with making multiple sweeps is that if there is any problem with transforming the scans into the point cloud, they will introduce messy outliers into the map, making it contain errors and incorrect features. It may be possible to reduce this effect with additional filtering before adding points to the final map.

**DISCUSSION AND FUTURE WORK**

There is always a tradeoff between cost efficiency and performance. Due to the fact that we are using low-cost sensors, we will inevitably need to compromise in performance. We are able to get a higher level of computer performance by picking specific components that have the right speed to meet our goals while still costing less than a comparable laptop.

Another approach under testing is to use multiple TortoiseBots cooperatively. We have built three identical robots and are currently configuring them to act as a distributed system, more powerful than any one robot could be alone. The questions we will look to answer are (i) where do we position these robots relative to each other to increase the total information, (ii) how do we control their angular speed and phase of rotation (if they are rotating), (iii) how do we move the robots altogether and (iv) who does the processing and what information do the robots share?

Having faster processing will leave overhead for additional algorithms such as more autonomous behavior and increases in performance. Because of processing limitations on the existing Turtlebot, three dimensional SLAM, known in this case as RGBD SLAM, is too intensive to work well onboard in real-time. The data is too large to stream to a powerful offboard computer so previously, all computation had to be done offline after the data was recorded. We should be able to run this algorithm on the TortoiseBot leveraging the extra processing power.
We plan to investigate the next-best-view problem in order to eventually make the entire scanning and mapping procedure autonomous. The next-best-view problem is the problem of finding the optimal position and orientation for the next measurement under different constraints [20]. The robot should also be able to decide when an area has been sufficiently scanned and move on. With these elements, the robot can explore the environment on its own and find the best way to generate a complete point cloud of the room or area of interest. Fusing the readings of both the Kinect and the LIDAR sensors will help with this problem. We can use the Kinect to determine in real-time whether some regions have more detail and require dense 3D scans and navigate the robot accordingly.

Additionally, the density and accuracy of the scans may be improved by fusing heterogeneous sensors such as these. Thus, we plan on writing custom navigation and obstacle avoidance algorithms to analyze a space and generate the corresponding map as quickly and efficiently as possible. Furthermore, the LIDAR is rigidly attached to the robot. This makes for a very simple design but limits the usefulness of the sensor. On other robots, the sensor is mounted to a precise mechanism for panning and tilting [2], [1]. In this way, the laser is swept without requiring the robot to move. Future versions of the TortoiseBot may feature the ability to rotate the LIDAR or at least turn it 90° from a vertical to horizontal orientation. This way the sensor may also be used for 2D localization and obstacle avoidance when not making maps. The 240° field of view gives a great overview of the 2D world. Finally, the on-board communications hardware of the TortoiseBots will enable us to build wireless networks among multiple robots, and be utilized for autonomous coordinated sensing tasks.

CONCLUSION

In this paper we have presented the design for a low-cost, 3D mapping, mobile robot platform. We used off-the-shelf components, some hardware modifications, fabrication, and open-source software to construct the TortoiseBot platform. Each component was chosen with regard to price, performance, and energy consumption. The design is also upgradable to match the mission requirements of the user. Our design goals of a target cost of less than the cost of a laptop with comparable performance and performance goal to use ≤ 25% of the systems processing power while running all the required hardware drivers and networking were both met. We demonstrated some representative workload tasks with the platform to showcase its capabilities.

REFERENCES


<table>
<thead>
<tr>
<th>Object</th>
<th>Scan Rate (°/sec)</th>
<th>Passes</th>
<th>Measured Value (cm)</th>
<th>Actual Value (cm)</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box 1 Width</td>
<td>10</td>
<td>1</td>
<td>22.00</td>
<td>23.00</td>
<td>4.35</td>
</tr>
<tr>
<td>Box 1 Width</td>
<td>10</td>
<td>2</td>
<td>22.50</td>
<td>23.00</td>
<td>2.17</td>
</tr>
<tr>
<td>Box 1 Width</td>
<td>10</td>
<td>3</td>
<td>21.34</td>
<td>22.00</td>
<td>7.22</td>
</tr>
<tr>
<td>Box 1 Width</td>
<td>10</td>
<td>4</td>
<td>20.98</td>
<td>22.00</td>
<td>8.78</td>
</tr>
<tr>
<td>Box 1 Height</td>
<td>10</td>
<td>1</td>
<td>12.45</td>
<td>13.30</td>
<td>6.39</td>
</tr>
<tr>
<td>Ceiling Height</td>
<td>7.5</td>
<td>1</td>
<td>270.56</td>
<td>270.00</td>
<td>0.21</td>
</tr>
<tr>
<td>Hallway Width</td>
<td>7.5</td>
<td>1</td>
<td>177.48</td>
<td>170.00</td>
<td>4.40</td>
</tr>
<tr>
<td>Box 2 Width</td>
<td>10</td>
<td>1</td>
<td>50.22</td>
<td>50.00</td>
<td>0.44</td>
</tr>
<tr>
<td>Box 3 Width</td>
<td>7.5</td>
<td>1</td>
<td>25.10</td>
<td>24.60</td>
<td>2.04</td>
</tr>
</tbody>
</table>

TABLE IV
EXPERIMENTAL RESULTS