Washington University in St. Louis

Washington University Open Scholarship

All Computer Science and Engineering Research

Computer Science and Engineering

Report Number: WUCSE-2010-71

2010

An Inexpensive Robot Platform for Teleoperation and Experimentation

Daniel A. Lazewatsky and William D. Smart

Most commercially-available robots are either aimed at the research community, or are designed with a single purpose in mind. The extensive hobbyist community has tended to focus on the hardware and the low-level software aspects. We claim that there is a need for a low-cost, general-purpose robot, accessible to the hobbyist community, with sufficient computation and sensing to run ``research-grade'' software. In this paper, we describe the design and implementation of such a robot. We explicitly outline our design goals, and show how a capable robot can be assembled from off-the-shelf parts, for a modest cost, by a single... **Read complete abstract on page 2.**

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation

Lazewatsky, Daniel A. and Smart, William D., "An Inexpensive Robot Platform for Teleoperation and Experimentation" Report Number: WUCSE-2010-71 (2010). *All Computer Science and Engineering Research.*
https://openscholarship.wustl.edu/cse_research/53

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

An Inexpensive Robot Platform for Teleoperation and Experimentation

Daniel A. Lazewatsky and William D. Smart

Complete Abstract:

Most commercially-available robots are either aimed at the research community, or are designed with a single purpose in mind. The extensive hobbyist community has tended to focus on the hardware and the low-level software aspects. We claim that there is a need for a low-cost, general-purpose robot, accessible to the hobbyist community, with sufficient computation and sensing to run ``research-grade'' software. In this paper, we describe the design and implementation of such a robot. We explicitly outline our design goals, and show how a capable robot can be assembled from off-the-shelf parts, for a modest cost, by a single person with only a few tools. We also show how the robot can be used as a low-cost telepresence platform, giving the system a concrete purpose beyond being a low-cost development platform.

An Inexpensive Robot Platform for Teleoperation and Experimentation

Authors: Daniel A. Lazewatsky and William D. Smart

Corresponding Author: dlaz@cse.wustl.edu

Abstract: Most commercially-available robots are either aimed at the research community, or are designed with a single purpose in mind. The extensive hobbyist community has tended to focus on the hardware and the low-level software aspects. We claim that there is a need for a low-cost, general-purpose robot, accessible to the hobbyist community, with sufficient computation and sensing to run ``research-grade'' software. In this paper, we describe the design and implementation of such a robot. We explicitly outline our design goals, and show how a capable robot can be assembled from off-the-shelf parts, for a modest cost, by a single person with only a few tools. We also show how the robot can be used as a low-cost telepresence platform, giving the system a concrete purpose beyond being a low-cost development platform.

Type of Report: Other

An Inexpensive Robot Platform for Teleoperation and Experimentation

Daniel A. Lazewatsky and William D. Smart

Abstract— **Most commercially-available robots are either aimed at the research community, or are designed with a single purpose in mind. The extensive hobbyist community has tended to focus on the hardware and the low-level software aspects. We claim that there is a need for a low-cost, general-purpose robot, accessible to the hobbyist community, with sufficient computation and sensing to run "research-grade" software. In this paper, we describe the design and implementation of such a robot. We explicitly outline our design goals, and show how a capable robot can be assembled from off-the-shelf parts, for a modest cost, by a single person with only a few tools. We also show how the robot can be used as a low-cost telepresence platform, giving the system a concrete purpose beyond being a low-cost development platform.**

I. Introduction

Most commercial robots fall into one of two distinct categories. Either they are aimed at the research market or at general consumers. Research robots tend to be prohibitively expensive (from the consumer point of view, at least), lack application-level software, and have a daunting learning curve (for those not already familiar with robotics). Consumer robotics products, on the other hand, tend to be inexpensive, but are often engineered for a single purpose, lack sophisticated computation resources and sensors, and are not really designed for tinkering with.

There is a large, active hobbyist robotics community, but much of their effort goes into designing and implementing the hardware and low-level control for their robots, frequently relying on low-power embedded microcontrollers for all computation. This limits the type of software development that is easily done on these platforms; the development environments and support libraries for these environments are typically not as rich as for more powerful, full-featured computing systems. For those with a software background, this can be seen as a limitation, making it hard to get involved with robotics.

By designing a robot with a full computer system, including an off-the-shelf operating system and a variety of development environments and tools, we hope to lower the barrier for entry for programmers, and those without a low-level software background. In addition, our choice of middleware for the system makes available a wide variety of common robotics algorithms, making it possible to develop application-level software more quickly than if one had to implement these fundamental algorithms from scratch.

In this paper, we present a robot system designed to bridge the gap between these three worlds: hobbyist, research, and commercial. Our goal is to allow *hobbyists* to experiment with *research*-grade software, on an accessible and extensible platform constructed from *commercial* off-the-shelf parts. In particular, we are interested in making it easy for those with only a software background to get involved in robotics at the hobbyist level.

Towards this end, we have designed and implemented a low-cost robot from off-the-shelf parts that can be assembled by a non-expert in a few hours. All of the parts are readily available from on-line sources, and we have provided extensive assembly instructions on our web site. The robot uses only open-source software, and can be running autonomously only a few minutes after the software is installed.

We also present an example open-source telepresence application for our robot. This application serves an example that exercises much of the functionality of the robot, gives the robot a concrete purpose, and provides example source code which can be used as a starting point for further software development.

II. Design Goals

Our overall goal is to bridge the gap between the research and hobbyist robotics communities, especially for individuals with a background in software rather than in hardware. Our design goals for this system are:

1) **Off-the-shelf parts.** The system should be assembled from commonly-available off-the-shelf parts, by someone with no formal hardware training.
2) **Easy to assemble.** The system should be simple enough to assemble that a 12-year-old can manage it under adult supervision. The assembly should only use commonly-available hand tools.
3) **Low cost.** The system should be affordable to the hobbyist community. Although "affordable" means different things to different people, we feel it is reasonable to define it as being of comparable cost to a desktop computer system.
4) **Useful.** The robot should be useful out-of-the box. Specifically, it should be more than a collection of parts and low-level software. In addition to being a platform that enables tinkering and experimentation, it should have a concrete purpose.
5) **Open and extensible.** The hardware and software for the system should be open, allowing users to modify it

Fig. 1. A full view of the robot.

shifts the interactions possible with the robot from looking at peoples' feet, to being able to look at their faces. This is important for our example telepresence application (see section IV-B), but does introduce some addition problems, mostly with the stability of the system. Should a user choose to do so, it is trivial to omit the stalk, and mount the netbook directly on the base, making the system much more stable (but limiting the applications that the system can be used for).

A. Specific Hardware Used

The major hardware components in the robot are:

1) **Base:** iRobot Create.
2) **Computer:** Lenovo S10-3t netbook. Any other similarly-sized convertible tablet computer could also be used.
3) **Pan/Tilt Camera:** Logitech Orbit USB camera.
4) **Navigation Camera:** Logitech QuickCam Pro 9000 USB camera. Many other low cost webcams could also be used.
5) **Body:** 80/20® T-slotted extrusion.

All of these are consumer-grade and consumer-priced, and are generally available on-line. While the specific models used are not important, especially for the cameras and computer, all of the hardware components must work reliably with Linux. USB cameras have traditionally not had strong Linux support, so some care must be exercised in the selection of the these devices, but USB video device class (UVC) cameras should work out of the box with modern Linux distributions.

The pan/tilt camera is used during teleoperation as the primary camera for the remote operator. It points out at the world, and can be moved to attend to different things. The navigation camera is mounted lower down the body, pointed at the floor directly in front of the robot. Studies by Paulos and Canny [9] as well as our own experiences suggest that the view from this camera is extremely helpful, allowing the remote user to more effectively navigate the robot through tight spaces.

In addition to providing a standard hardware configuration, in keeping with our goal of an *open and extensible* system, the platform should be open enough so it can be easily extended and tailored to user needs. The use of 80/20® hardware allows use of an extensive catalog of inexpensive and easy to assemble parts, as well as simple mounting of cameras and other devices with standard tripod mounting holes. A full parts list can be found on our website[1] along with several recommended configurations.

B. Limitations of the Hardware Design

Assembling the robot from off-the-shelf parts has resulted in a number of limitations. Although solutions to several of these limitations are straightforward, most involve access to tools not commonly found in the home, or assume a certain level of skill with hardware. While we expect some users

as they please.

6) **Accessible to non-experts.** It should be possible to run the system, and to make modifications to it, without an advanced degree in robotics.
7) **Capable.** The robot should be capable of running "research grade" robotics algorithms, of the sort that appear in our conferences and journals. These algorithms should be available for users to experiment with, even if they do not have a full understanding of how they work.

III. HARDWARE

The robot, shown in figure 1, consists of an iRobot Create mobile base with a stalk, approximately 1.22m in length, extending upwards. A convertible tablet netbook is clamped at the top of the stalk, providing all of the robot's computing power, along with speakers and a microphone. A pan-tilt camera is mounted above the computer.

In designing a robot that is useful and capable, we would like the robot to be able to interact with people and things in a human-scale world. This means that simply placing a netbook on an inexpensive mobile base (such as the iRobot Create) is not sufficient; the stalk mounted on top of the base

---

[1]http://telepresence.cse.wustl.edu

of this platform to possess both the skills and ability to overcome these limitations, we cannot make this assumption for all of the intended users, many of whom will have no such tools or expertise.

The most significant limitation is in the stability of the robot. Mounting the laptop high above the base places it at approximately at (short) human face height, but also makes the system somewhat unstable, since the laptop is heavy compared to the rest of the components. This can be addressed by putting extra ballast in the base (extra batteries, for example), or by imposing software limits on acceleration (see section IV-C).

Flexing in the plastic base is also a source of instability. This flexing is actually a feature of the Create base, since it allows it to navigate over the edges of carpets and other slightly raised obstacles. The base could be stiffened with additional hardware, but this would come at the cost of reduced mobility. Again we made the decision to retain the mobility, and deal with the stability issues in software.

IV. SOFTWARE

A. Software Architecture

The robot control software uses ROS [10], a free, open-source middleware platform for robotics. ROS meets our design requirements of openness and extensibility and low cost, and also provides the following important features:

1) **Support for multiple programming languages.** At the time of writing, ROS supports C++, Python, Octave, and LISP. Python support is particularly important, since this is a popular, accessible language that is easy for non-specialists to learn and use.

2) **Well-documented, and with a supportive user community.** ROS has extensive on-line documentation, a mailing list for questions, and a large, supportive user community. This is a vital resource for hobbyists who do not have access to a "local expert" to help them with software problems as they come up.

3) **Access to research-grade robotics algorithms.** There is an extensive and constantly-growing collection of robotics software available in ROS. At the time of writing, over thirty research institutions are actively publishing ROS software. This gives users access to a wide variety of state-of-the-art algorithms for path-planning, localization, mapping, and SLAM.

4) **Extensive suite of development tools.** In addition to the middleware itself, ROS has a number of visualization and development tools that make it easier for novice programmers to develop and debug robot control software.

5) **Modularity.** ROS is designed to be highly modular, allowing users to largely ignore the parts of the control software that they are not interested in (low-level drivers, for example), and to concentrate on the ones that are important to them. It also allows algorithms and modules to be portable across robot platforms, as long as the low-level driver modules are in place, and expose a common API.

The ROS middleware is structured as a graph of independently-running nodes, each of which publishes the results of some computation on incoming data, either from sensors or from other nodes. We refer the interested reader to the paper by Quigley *et al.* [10] for more details on ROS, but note that it is a highly modular system, which allows high-level applications to be constructed relatively quickly. Nodes for many common robotics algorithms (path-planning, navigation, mapping, SLAM, *etc.*) are already available, documented on-line, and integrated into the middleware

B. Telepresence Application

The primary goal of this work is the design and implementation of a low-cost robot that will make it easier for those with a software background to become involved in robotics. However, just providing a hardware platform leaves a potential user with two important questions: (1) What is it good for? (2) What do I do now?

To address these questions, we have also developed an example telepresence application for our robot. This serves to give the robot something that it can do "out of the box," while also providing an example application that exercises all of the hardware on the robot.

Robot telepresence systems are becoming increasingly popular [3] (see section VI for an overview) and are something that a potential hobbyist will likely be familiar with, at least in concept. As a telepresence system, We do not claim that our system is a as capable as many of the commercial solutions; our goal is to use the telepresence application to illustrate what our robot can do, how to write applications for it, and to provide an example application. Having said that, the application is sophisticated enough to use in a realistic setting. The authors have used it to successfully participate in remote meetings, including at doctoral thesis defenses.

The software for the telepresence application is built on the ROS middleware, and is available for download from our web site. It includes all of the drivers required for interacting with the base and cameras, and a web-based interface for remotely driving the robot about. The web interface is shown, shown in figure 2 is loosely based on Willow Garage's Texai web interface, and implements simple keyboard-based controls for moving the robot. The full system is shown in figure 3, and consists of the following custom nodes:

1) **iRobot Create drivers**. Developed by Brown University[2], this node takes in ROS velocity command messages, and exposes iRobot Create's sensor data in the ROS message format.

2) **Webcam driver**. Uses OpenCV to capture images from the two webcams and send them as ROS messages [2].

3) **Image saver**. Listens for webcam images and makes them available to the web interface.

4) **Driving web application**. Written as an application for ROS' `webui` package, this presents a HTML interface which listens for keypresses, sends keypress data to
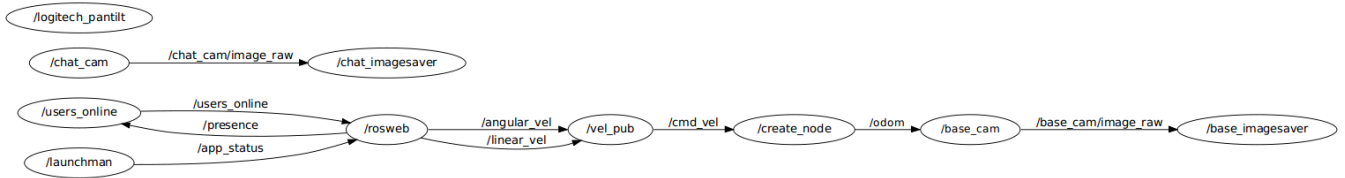
---

[2]`http://code.google.com/p/brown-ros-pkg/`

Fig. 3. A screenshot of `rxgraph`, one of the many debugging tools included with ROS showing the node graph of the included software stack.
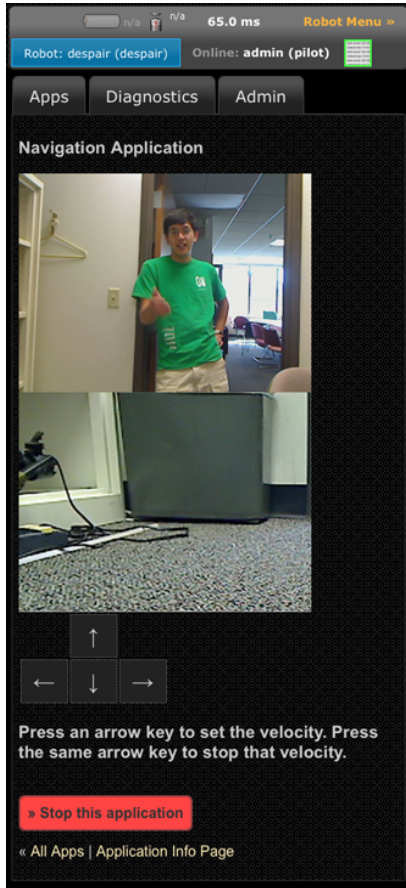

Fig. 2. The basic web interface, showing a keyboard-based driving interface, and view of the two cameras.

ROS using HTTP requests, and translates those HTTP requests into ROS velocity messages.

C. Overcoming Hardware Limitations in Software

The use low-cost consumer-grade hardware, while lowering the overall cost of the system, has the effect of lowering the robustness and quality of the parts used, compared to research-grade robots. However, many of the limitations introduced by inexpensive hardware can be compensated for in software.

1) Inaccurate Pan/Tilt Movement: The pan/tilt camera used on the robot is unable to report its pose, and cannot be accurately and consistently commanded to a given pose. This uncertainty increases with time, as the nominal pose and the actual pose diverge more and more. This problem

is likely due to slippage and backlash in the gear train and to inaccurate motor encoders. These errors can be modeled, and the current pose of the pan/tilt mechanism tracked with an Extended Kalman Filter. Driving the unit to its pan and tilt limits on initialization will give us a known starting pose.

2) Unstable, Lightweight Base: The instability of the robot can be decreased by simply adding extra weight to the base. However, this makes the robot heavier, slows it down, and shortens the battery life. We addressed this problem by putting in place software limits on the robot's motions. With no limits, the robot accelerates to its maximum velocity of approximately 0.35 ms^{-1} almost instantly. If we assume an acceleration period of 0.5s, this is an acceleration of 0.7 ms^{-2}.

We placed asymmetric limits on acceleration and deceleration. Empirical testing showed that a maximum acceleration of 0.6 ms^{-2} and maximum deceleration of -1.0 ms^{-2} resulted in stable operation. Velocities are constrained such that

$$v_t = v_{t-1} + a \, \mathrm{d}t$$

until the target velocity is reached, using a controller frequency ($\mathrm{d}t^{-1}$) of 20Hz.

The acceleration limits made the robot significantly more stable, and difficult (though not impossible) to tip over accidentally. The acceleration values were chosen to be as large as possible to minimize any perceived sluggishness in the controller.

3) No Depth Sensor: Most research robots have depth sensors, such as laser range-finders or stereo vision systems, available to them. However, these devices tend to be expensive, either to purchase or in terms of computation. We can give our robot some basic range-sensing capabilities by implementing Horswill's Polly algorithm which uses a single, uncalibrated, low-resolution camera to do visual obstacle avoidance [4]. The algorithm assumes that the area directly in front of the camera is free space, and builds a color histogram model of it. All other areas in the image that match this model are assumed to also be free, and are marked as safe. Horswill observed that when the camera is positioned near the floor, approximately parallel to it, pixel height in the image corresponds to depth in the world. The Polly algorithm takes advantage of this by searching from the bottom of the image up, looking for the highest 'safe' pixel in a contiguous block, and moving towards it.

The Polly Algorithm has been implemented on a 75 MHz embedded microcontroller [12], and can be expected to run well on the Atom processor of our robot's netbook.

Research grade depth sensors (stereo cameras, laser rangefinders, structured light cameras, time of flight cameras, etc) are extremely expensive, generally costing more than our complete robot. This makes autonomous obstacle avoidance more difficult, and limits the use of many current SLAM algorithms. The sensors the robot does have lend themselves well to use with visual odometry (vSLAM) approaches, such as vSLAM [6] and MonoSLAM [1]. However, these tend to be computationally expensive, and are unlikely to be able to run on the relatively modest processors on most netbook-class computers. However, as netbooks get more powerful, it may be possible in the future. Additionally, as low cost depth sensors (such as the Microsoft Kinect [7]) become available, the extensible design of our platform will allow for their easy integration.

4) Autonomous Docking: In addition to causing instability, we found that the added weight caused the Create to move too slowly to dock autonomously with its charging station when using its built-in docking routines. When loaded with our additional hardware, the robot cannot build sufficient momentum to properly insert itself into the dock.

To overcome this, we have implemented our own autonomous docking routine, which exposes the robot's speed as a user-defined parameter. The routine uses the Create's infrared sensor to position the robot in front of the dock and drive along a path until it is properly docked.

V. USER EXPERIENCE

Our goal is to make the process of acquiring the parts, building the robot, and running the demonstration application to be as simple and straightforward as possible. We believe that early frustration with the system might cause some potential users to give up before they have a working robot, which we want to avoid at all costs.

To that end, we have created an on-line parts listing, integrated with a wish-list at `amazon.com`. This means that potential users can order the parts for their robots quickly, with only a few mouse clicks. Once the parts arrive, users can download detailed assembly instructions from our web site. Informal testing indicates that assembling the system from parts, using the instructions, and with no additional help takes less than two hours for someone unfamiliar with the robot. The entire robot can be assembled using only two hex keys (included in the parts list).

Once the hardware is assembled, the user must install Ubuntu Linux on the netbook, again following on-line instructions. Once the operating system is in place, the ROS middleware can be installed using the Ubuntu package manager (just like any other set of software packages). We include detailed instructions for this on our web-site, assuming no prior knowledge of Linux. We also give links to other web sites, showing how to configure and customize the Linux installation. Our future plans include a custom Linux distribution, with all necessary components pre-installed, that users can download directly from our web-site.

Once the operating system and ROS middleware are in place, executing a single script will start the web interface, which can then be accessed from other computers via a web browser. Again, we provide instructions that assume no prior knowledge of Linux, with pointers to assist in debugging network-related issues.

To provide further assistance, we have established a mailing list and wiki on our web-site, for users to ask (and answer) questions about installation, and other aspects of the system.

A. Software Development

Although an introduction to computer programming is beyond the scope of our goals, we do provide links to introductory resources on our web-site. We also provide links to specific sections of the ROS on-line documentation, and tips on how to become a member of the ROS user community. The ROS documentation includes an overview of how ROS works and the concepts behind it, as well as a set of tutorials covering the basics of interacting with ROS such as navigation the ROS filesystem, creating and building package, and using ROS' included tools, through writing nodes and subscribing to topics. All of the basic tutorials which involve writing code have versions for both C++ and Python.

After users have completed the ROS tutorials, our telepresence application provides a concrete starting point for developing software applications for the robot, and interacting with its hardware. The modular design of ROS makes is particularly easy for the components of the telepresence application (and any other application, for that matter) to be reused as the building blocks of something new.

VI. RELATED WORK

There have recently been several entries in into the commercial telepresence market, most notably the Texai from Willow Garage, Inc, the QB from Anybots, Inc and the VGo from Vgo Communications. These systems, though all very capable, are prohibitively expensive for consumers and hobbyists, many costing upwards of $10,000 USD [5] [3]. On the other end of the spectrum, there are numerous hobbyist-grade telepresence projects, including Sparky Jr.[3], an approximately $1,100 USD iRobot Create based telepresence robot using custom hardware and software, and other more expensive custom platforms. There are also hobbyist-grade robots like LabRat[TM] [11], which was designed to be an extremely inexpensive robot for classroom, hobbyist and research use. Unlike these robots, we are not trying to provide a commercial telepresence platform, or a super-low cost hobbyist robot, but rather a capable, extensible development platform which can be configured for telepresence purposes at a cost of an order of magnitude less than what is currently available in similarly capable robots.

In 1998, Paulos and Canny asserted "[personal tele-embodiment] systems can now be built at minimal cost" resulting in "the extension of robotics in to the lives of ordinary people," [8] but no viable options have yet to materialize. If telepresence (and by extension, personal robotics)

---

[3]`http://http://sparkyjr.ning.com/`

are to permeate society as stated, the necessary understanding of how to design effective software systems and interfaces must progress significantly, and be available to hobbyists and early adopters because, as we claim, the critical mass of users does not exist within the research community for this to happen. This critical mass can only come about if the hardware necessary for basic work is within reach of anyone wishing to experiment, and if the work done by these people can be easily shared.

VII. Conclusions and Future Work

We have introduced the basic hardware requirements for a functional telepresence robot, and put forth a low-cost design that meets these requirements. This platform moves towards bridging the gap between hobbyist and research robotics by providing a standard platform, along with research-calibre software development framework, and the ability for hobbyists to take advantage of cutting edge systems in their own work. The hardware is no more difficult to assemble than the furniture bought by millions of people each year, and the software can be installed automatically with several scripts. We hope that this platform can put a low-cost, easy to assemble, and useful robotic platform in the hands of anyone interested in writing robot software, and provide a basis for meaningful hobbyist involvement in the open-source robotics community.

In building a low-cost robot, we must forgo using research-grade parts, instead using consumer-grade hardware which does not perform as well as the more expensive alternatives, described in more detail in section IV-C. It is desirable for these inexpensive parts to have performance better than was intended by design. Out of the box, the pan tilt camera used gives no estimates of its state (pan and tilt angles). We have begun work on using visual observation sources to provide state estimates to be used in Sparse Bundle Adjustment or a Kalman Filter.

Given a low-cost, easy to assemble robotics platform, we hope to hold a robot-building workshop at the St. Louis Science Center to show children that even though the world of robotics is complex and can be overwhelming, getting started is easy. We plan on making these robots available to operate over the internet using a web-based interface, allowing people all over the world to explore the museum, and providing a testbed for further human-robot interaction work by ourselves and other HRI researchers.

Acknowledgements

References

[1] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:1052–1067, 2007.

[2] Adrian Kaehler Gary Bradski. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2008.

[3] Erico Guizzo. When my avatar went to work. *IEEE Spectrum*, September 2010.

[4] Ian Horswill. Analysis of adaptation and environment. *Artificial Intelligence*, 73:1–30, 1995.

[5] Anybots Inc. Anybots - your personal avatar. `http://anybots.com/#qbLaunch`, August 2010.

[6] D. Krys and H. Najjaran. Development of visual simultaneous localization and mapping (VSLAM) for a pipe inspection robot. In *Computational Intelligence in Robotics and Automation, 2007. CIRA 2007. International Symposium on*, pages 344 –349, jun. 2007.

[7] Microsoft. Xbox.com — kinect. `http://www.xbox.com/kinect`, September 2010.

[8] E. Paulos and J. Canny. Designing personal tele-embodiment. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 4, pages 3173 –3178 vol.4, may. 1998.

[9] Eric Paulos and John Canny. Social tele-embodiment: Understanding presence. *Auton. Robots*, 11(1):87–95, 2001.

[10] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[11] P. Robinette, R. Meuth, R. Dolan, and D. Wunsch. Labrat^{TM}: Miniature robot for students, researchers, and hobbyists. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009)*.

[12] A. Rowe, C. Rosenberg, and I. Nourbakhsh. A low cost embedded color vision system. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 208 – 213 vol.1, 2002.