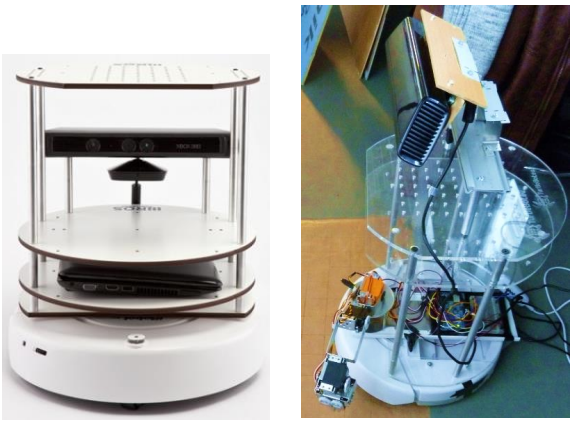
A low-cost custom-built robotic arm on the TurtleBot platform

Pierre Herman*, Alejandro R. Mosteo†‡, Ana C. Murillo*‡
*Dpt. de Informática e Ingeniería de Sistemas. University of Zaragoza, Spain
†CUD - Centro Universitario de la Defensa. 50090 Zaragoza, Spain
‡I3A - Instituto de Investigación en Ingeniería de Aragón. 50018 Zaragoza, Spain

***Abstract*—This document presents a project that covers all the necessary steps to build a low-cost mobile platform equipped with a robotic arm. A Kinect-guided custom robotic arm on the TurtleBot platform provides the vacuum cleaner with basic manipulation capabilities. All stages in the project involve standard hardware components and open-source software modules. 1) The robotic arm is built with off-the-shelf parts and controlled from an Arduino board. 2) A Kinect sensor is used for object detection. 3) Two ROS nodes manage all the elements, commanding the mobile base and defining targets to be grasped.**

The arm has been evaluated in order to identify the accuracy that can be expected with this setup, and a demonstration program, which simulates a simple cleaning task in which soft objects are moved out of the way, is described. The setup and algorithms used are simple and easy to replicate, and the experiments show notable robustness and performance from the system, making this set up useful for educational and research activities. This initial prototype has a simple pincer that enables grasping of easy objects only, while future work will address improvements for more advanced recognition and grasping strategies.



(a) (b)

Fig. 1. (a) Original TurtleBot platform. (b) Prototype built in this project with a robotic arm.

I. Introduction

Service robots and their components are becoming a commodity in recent times, with growing offerings and cheaper opportunities. This can be seen in robotic bases, e.g., autonomous vacuum cleaners [5]; in micro-computers that can provide the necessary computing power like the many ARM-based boards [4, 14]; and in of-the-shelf parts that enable rapid prototyping[15]. This project leverages these factors to augment a standard vacuum cleaner base with a custom arm capable of grasping simple items, like small pieces of clothing, that are in the way of the robot. Lessons learned and solutions achieved may be of interest as an affordable project for courses on robotics, vision, grasping and related subjects.

The project scope was to build a custom arm and integrate it with a TurtleBot base [3], an open low cost robotic platform. Figure 1 shows the original TurtleBot platform and the final version built after integrating the robotic arm. The goal is to allow the robot to identify objects in its path and, if they were suitable targets, remove them using the arm, in order to proceed with an hypothetical vacuum-cleaning task. Following the goal of using low-cost and commercial components, the perception that guides the grasping was carried out with the Microsoft Kinect[1] available on the TurtleBot platform. Kinect and other RGB-d sensors have an increasing presence in multiple robotics applications and platforms. Some applications are closely related to the initial goals of this sensor, such as human pose detection [13] and Human-Robot interaction [11]. They have also been used, with excellent performance, for plenty of other vision based tasks, such as visual SLAM [7] or, as we do in the presented project, combination of manipulation and vision tasks [8].

The main subtasks covered in this project and detailed in the following sections are: 1) Design and construction of the arm. Several choices were considered, from custom-built pieces to off-the-shelf blocks. 2) Inverse kinematic model for final positioning of the arm given a target in robot coordinates. Arm control from an Arduino board. 3) Vision module for target detection and localization from the point cloud readings of a Kinect sensor. 4) Prototype integration, including a basic movement behavior integrated with the rest of features, by means of ROS [10], and evaluation of the system performance, regarding accuracy on the arm positioning and the recognition capabilities.

Related projects can be found in the web[2],although they tend to be scarce in written details besides the parts used. Some of them go further in particular aspects, like manufacturing of the parts or remote control set up. Our project emphasizes the construction of the arm and vision-based target localization.

[1]http://www.xbox.com/en-US/kinect/

[2]http://www.youtube.com/watch?v=xrOhlBN-_ao http://www.orbduino.com

Model	Torque	Intensity
Hitec Hs-311 (x2)	3kg/cm	150mA
Hitec Hs-645	9kg/cm	350mA
Futaba S3003	3.2kg/cm	120mA
HBKing	9kg/cm	340mA

TABLE I
SERVOMOTORS USED IN THIS PROJECT

II. BUILDING THE ROBOTIC ARM

This section covers the design and construction of the robotic arm and its controller.

A. Components

An earlier prototype was built with custom wooden pieces and standard commercial servomotors, but it showed insufficient rigidness which resulted in increased imprecision. Aluminium seems a better choice thanks to its combination of lightness and stiffness and is the material considered for links of the current prototype.

1) Joints: To build the arm joints, standard servomotors were available for the project. A servomotor allows the positioning of a joint at a precise angle, which the servo then maintains thanks to its integrated feedback controller [1]. These actuators are commonly used in small robots [16], model airplanes, etc. In order to transmit the desired angular position, variable-length pulses have to be transmitted via the control wire. The servomotors used in this project are detailed in Table I.

2) Links: To build the links of the arm, we explored two possibilities that could make use of the servomotors available for the project: custom-made parts, bended and cut out from aluminium sheets, and ready-made kits of standard parts.

For the **custom-made** option, design objectives were lightness, sufficient stiffness and compatibility with standard servos. The pieces were designed with the CatiaV5 CAD software, which allows the necessary characterization of the parts, like center of gravity and equivalent mass, data which are necessary to ensure that the required motor torques are inferior to those provided by the servos. Finally, from the 3D design, 2D plans for fabrication are automatically generated, which were sent to several manufacturing firms.

Secondly, we explored the use of **standard kits** compatible with the servomotors. These kits [15] are composed by modular bricks that fit easily with each other. Besides, they are specially designed to be mounted on the servos minimizing play in undesired directions. Figure 2 shows how one of these bricks looks like. Further advantages are that there is no need for any fabrication of parts and there is the certainty that the bricks will properly fit as expected.

After evaluating both options, there was no clear price advantage for one of them (around 200€ to build the arm with each of them). However, using the commercial modular bricks provided a faster solution (manufacturing custom parts would take much longer than ordering the commercial bricks) and removed any uncertainty about unforeseen defects in the parts. Fig. 3 shows the finished prototype built with the chosen alternative.

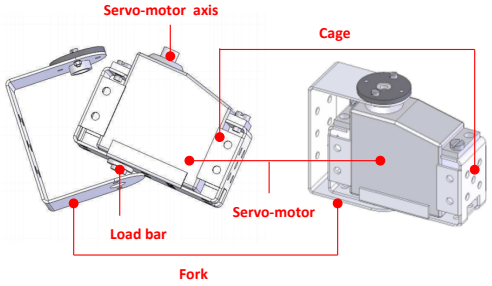


Fig. 2. Diagram of the bricks used to mount the arm.

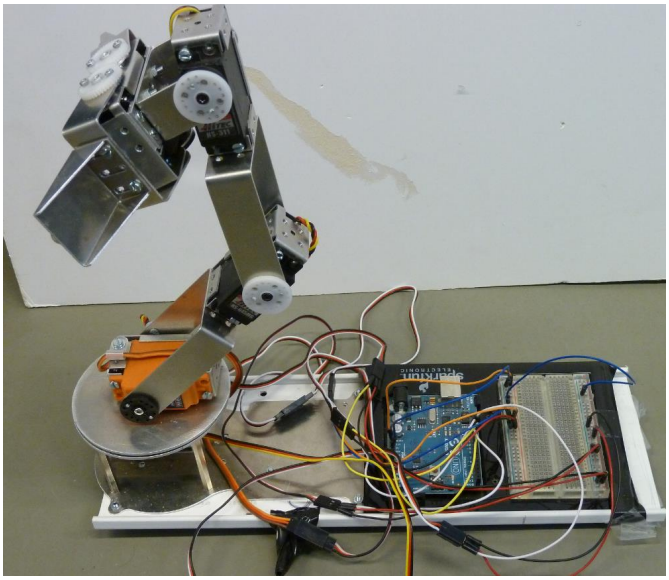


Fig. 3. Arm built with standard bricks, before being mounted on the robot.

B. Motion Control

In order to generate the pulses required by the servomotors, some kind of microcontroller is necessary. Due to previous experience, availability and affordability, an Arduino One microcontroller was used (see Fig. 4). It provides several input/output ports for digital and analog interaction. One drawback of this solution is that the combined power requirements of the motors was larger than the power output of the Arduino. The solution was to use the power line of an USB connector attached to the laptop running the high-level control software. Programming was done using the standard Arduino development environment, which relies on the C language.

An essential step to get the arm ready to work is to compute the inverse kinematic model of the arm. We use the DH (Denavit-Hartenberg) parameters to solve it [9]. This model allows us to compute the arm trajectory required for a desired position of the arm tool [6].

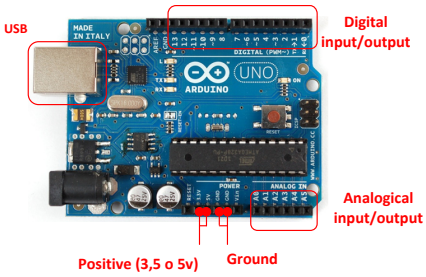


Fig. 4. Arduino One microcontroller used to control the arm motion.

C. Calibration

There were several aspects involved in the servomotor control that required calibration.

Firstly, the servomotor manufacturers promise a lineal proportion between pulse duration and angular position, but do not provide their relation. A contraption was devised (Fig. 5) in order to obtain such relations. A program communicating with the Arduino board was also programmed that sent pulses of a desired duration and collected (via manual input) the corresponding output angle. With those data the correspondences shown in Fig. 6 were obtained. Due to the the different responses, a conversion function had to be implemented for each servomotor.

A second aspect was that the servomotors do not provide tachometer feedback. Consequently, the initial angles corresponding to the zero configuration of the kinematic model could not be measured directly from them. A reverse procedure in which each joint angle was incrementally modified until the base position was reached was used to determine the servo angles for the arm rest position.

D. Control API

A number of procedures were implemented for arm command. These are listed in Table II. As can be seen, a complete (even if small) high-level API is available to programmers desiring to use the arm.

API call	description
`Init`	Resets the arm to its starting position.
`Calculate`	Computes joint angles for a target pose.
`Pose`	Absolute angular positioning.
`Move`	Joint movement with speed control.
`Moves`	Linear movement with speed control.
`Open`	Opens the pincer.
`Close`	Closes the pincer.

TABLE II
COMMAND API FOR THE ARM

III. OBJECT DETECTION AND GRASPING

The use case in this prototype is to detect small objects abandoned on the floor that might impede the robot vacuuming task. This being a proof-of-concept application, we only consider targets easy to grasp, that do not require very accurate grasping points to succeed.

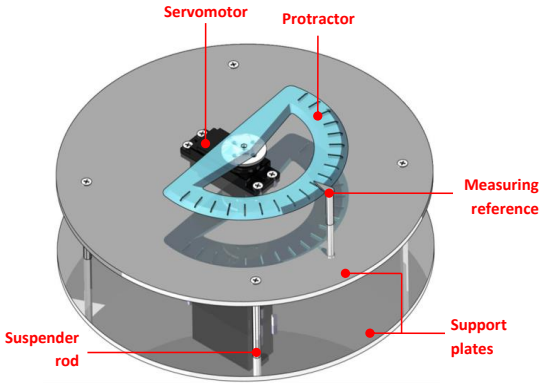


Fig. 5. Custom device for servo calibration.

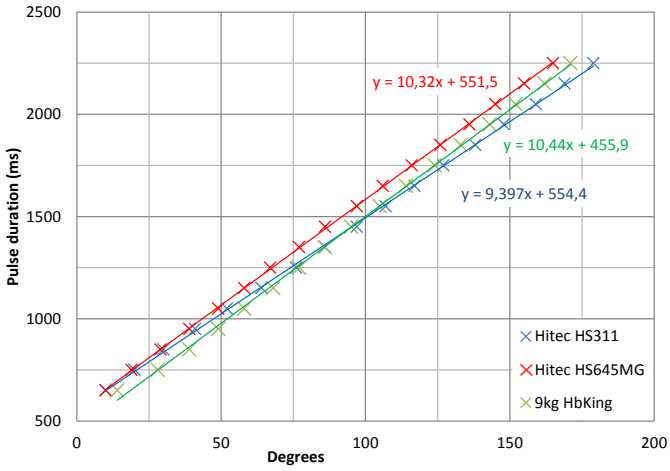
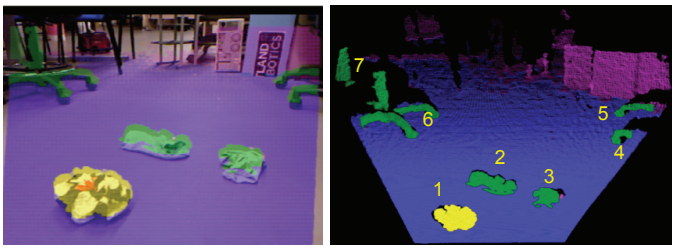


Fig. 6. Pulse duration to angle relations.

The vision sensor is a standard Kinect one, which provides 3D point clouds. The following steps were used in order to detect targets and estimate the grasping point where we need to direct the tool of the arm. Most of them are based on functions provided with the PCL library [12]:

1) Cloud voxelization: the voxel-grid [12] algorithm was used to speed up computations, given that the amount of pixels in the returned cloud were too large for acceptable processing time. This algorithm discretizes the space into small volumes or boxes, and all points within the same box are just represented with one point in the cloud. This process is similar to pixelization but in 3D instead of 2D.
2) Floor detection: given the reasonable assumption that the robot is operating in an indoor, flat environment, the floor plane was robustly detected and removed from the cloud by using the *SACSegmentation* [12] with RANSAC [2]. Figure 7(b) shows an example where the floor plane is segmented out (in blue) from the rest of the point cloud. The algorithm implemented considers that the dominant detected plane could be a wall; in that case the algorithm will remove those points and iterate again until an horizontal plane is processed.



(a) (b)

Fig. 7. Floor plane detection. (a) RGB image overlaid with the segmented point cloud. (b) Floor plane in blue, detected objects in green (each number indicates a different candidate object cluster), other points in magenta. The detected object considered as next grasping target is highlighted in yellow.

3) Object detection: following the idea of segmentation by proximity, this step runs an algorithm for Euclidean Cluster Extraction [12], which implements a K-mean clustering. Basically, any two points closer than a configurable threshold are considered to belong to the same object. An example of the object segmentation obtained can be seen in Fig. 7, where each of the numbered green clusters represents one cluster found in the point cloud that is a potential object.
4) Target selection: among all the detected objects in the previous step, one has to be chosen for grasping. The centroids of all objects are computed and the closest one to the robot is considered the target of choice, if it is within arm reach and its size is within an established interval. In the example of Fig. 7, the chosen object is represented in yellow, and corresponds to the closest object to the robot.
5) Conversion of coordinates: finally, the object coordinates are converted to arm coordinates in two steps, as described in Fig. 8. Firstly, the floor plane detected in step 2 is used to correct the camera rotation. Secondly, the known robot size and location of the vision sensor within the robot are used to apply the necessary translation 8.

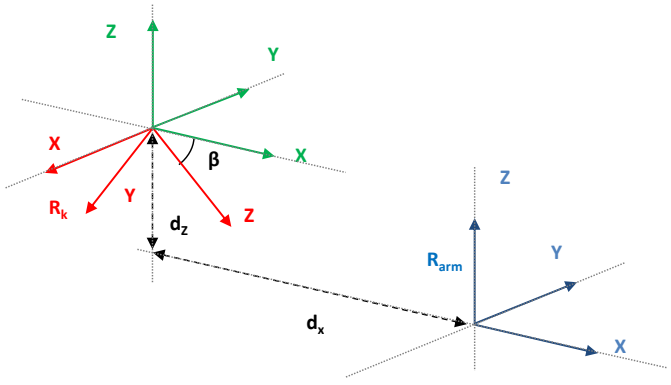


Fig. 8. Coordinate conversion. R_k is the Kinect reference frame, R_{arm} is the arm one.

IV. ROS AND PROTOTYPE INTEGRATION

This section discusses the final integration of the arm, vision subsystem and mobile platform. The robot is a standard *TurtleBot* platform from *Willow Garage* [3], easy to program and built upon a *Roomba* base without the vacuuming components. Using the described elements, a simple behavior was implemented in which the robot wanders around until an object is within reach, at which point it is picked up and tossed aside. The robot continues with this behaviour in a loop and if the bumper detects an obstacle it just changes the direction of its motion.

The subsystem integration was achieved with the *publish & subscribe* paradigm of ROS. In ROS, a so-called *node* is a stand-alone process which communicates with other similar processes by means of *topics*. A topic is a stream of data of a fixed and predetermined type. Interested parties can subscribe to read the data stream and, in turn, can create their own topics where they publish its own produced data. This way, a graph of information flows is formed 9.

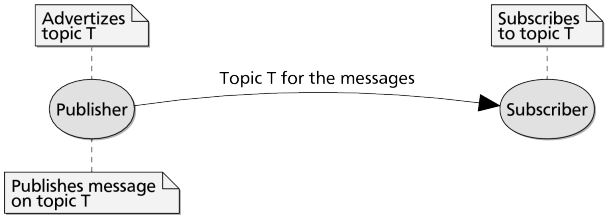


Fig. 9. A generic publish-subscribe example using ROS.

In our system, two nodes run on the robot, as shown in Fig. 10. The topic from the vision node publishes the coordinates of the cluster centroid from the closest object/target detected. The main node communicates with the Arduino controller and, if the arm is ready and a target is within reach, performs the grasping action. While there is no grasping action running, the main node communicates with the mobile base platform to generate the exploration motion.

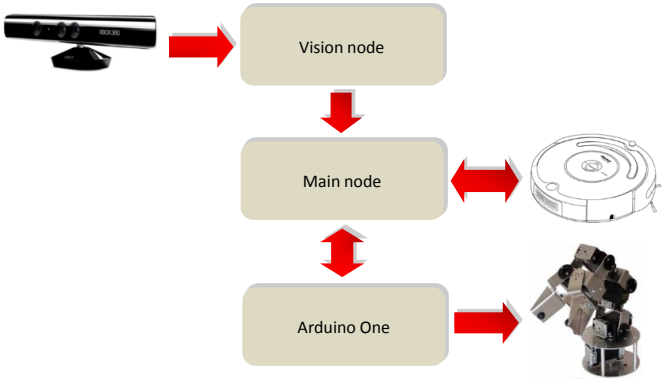


Fig. 10. Hardware and code interactions at the designed system on the robot.

Subsystem	Magnitude	Min	Max
Vision System	X (mm)	0	1200
	Y (mm)	−350	350
	Z (mm)	−170	200
Area of Detection in the ground	X (mm)	10	220
	Y (mm)	−170	170
	Z (mm)	−170	0
Arm Reachability	Range (mm)	10	300
	X (mm)	0	300
	Y (mm)	−300	300
	Z (mm)	−170	300
Target Size	Size (voxels)	90	2000
	Size (cm^2)	15	350

The origin of coordinates considered is at the arm base.

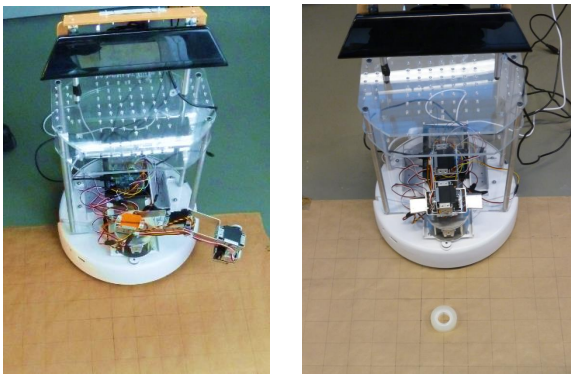
TABLE III
SUBSYSTEM OPERATING RANGES

A. Practical issues

A noticeable difference between the standard TurtleBot and our prototype is that the Kinect is differently mounted. In the former, Fig. 1(a), the sensor is placed at the rear of the robot, with horizontal aiming, in order to have a good perception of the environment in front of the robot. In our case, Fig. 1(b), a tilted position was chosen that permits capturing the immediate space at the floor level in front of the robot, with good definition in the closest surroundings (the exact ranges are shown in Table III). The determining factor for this configuration is the minimum range imposed by the Kinect sensor, blind at very short distances. The mounting was built with a few pieces of scrap metal and wood.

B. Ranges of operation

As part of the evaluation of the final prototype, we measured the several ranges of operation of each subsystem, which in turn determine the capabilities of the whole integrated system. A summary of these results is shown in Table III. Besides, a battery of tests was performed to evaluate the accuracy of the arm positioning and the vision subsystem (target localization), as detailed in the next subsections.



(a) (b)

Fig. 11. (a) Robot over the testing grid, which is centered under the origin of the arm coordinate frame, during the arm positioning experiments. (b) Robot over the grid, with the testing target shown, during the vision based target localization evaluation.

	Mean offset	Mean error	Min	Max	Std. deviation
X	0	2.3	−10	5	3.7
Y	−0.4	3.9	−10	10	5.1
Z	8.4	8.5	0	25	7.0
d	2.8	4.1	−6	10	4.1

TABLE IV
ARM POSITIONING ERROR (IN MM)

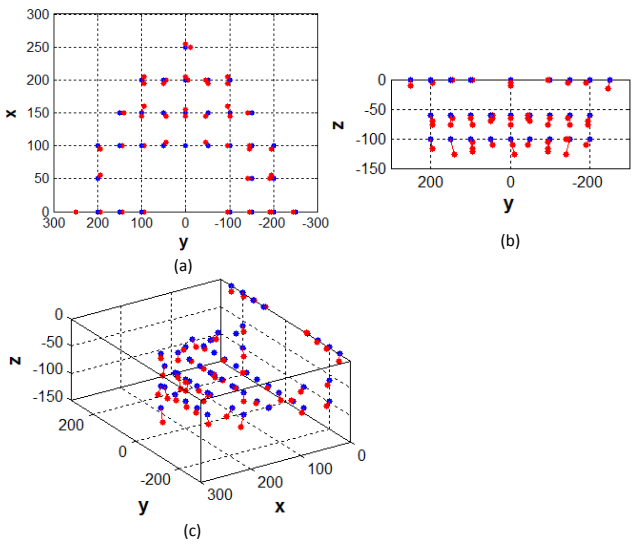


Fig. 12. Arm positioning measurements. Target values are in blue, actual arm position in red. (a) Top view. (b) Front view. (c) Perspective.

C. Arm positioning accuracy

In this experiment we measured the error between a reference position command and the actual final position of the arm. A grid was devised that facilitated the measurements (Fig. 11). Averaging over 50 runs for each axis, we obtained the results summarized in Fig. 12 and Table IV. Two conclusions can be drawn:

- Errors in the Z axis are larger, which can be attributed to the gravity influence on the servos position.
- The larger the distance, the larger is the error. This is consistent with the previous observation, since the torque exerted by gravity on the servos grows with distance.

It is presumable that better servos would improve the Z error, which is the only significant one, being close to one centimeter in the worst case.

D. Vision-based target-localization accuracy

A similar experiment (Fig. 11 (b)) was conducted in which we evaluated differences between reported target centroids and ground truth positions of the target object (obtained manually by using the same grid as in the previous test). Averaging 50 runs, we obtained the results shown in Fig. 13 and Table V. We highlight two salient observations:

- There is some correlation between the errors in Z and Y, suggesting that the camera was not perfectly aligned to the ground plane.

	Mean offset	Mean error	Min	Max	Std. deviation
X	−2.3	2.5	−6	1	2.5
Y	−4.1	4.1	−8	−1	2.2
Z	2.1	2.0	−1	7	2.2

TABLE V
TARGET LOCALIZATION ERROR (IN MM)

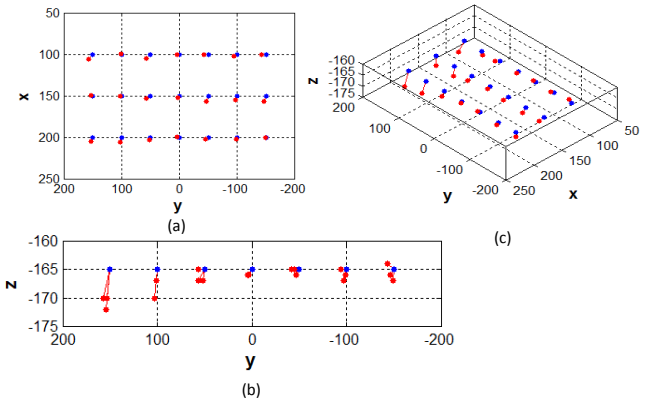


Fig. 13. Target detected coordinates. Ground-truth values are in blue, computed ones in red. (a) Top view. (b) Front view. (c) Perspective.

- Errors in Y are constant and negative, suggesting that coordinate conversions between Kinect and arm references could be improved.

These errors are quite small but could be decreased by further adjustment of the transformation function and camera alignment. Besides, it is possible that some error was introduced in the measurement of the ground truth, via imperfect placement of the target, which was performed manually.

V. CONCLUSION

This document described the design, building, integration and testing of a small and affordable custom arm with a pincer that can be of interest in teaching and initiation to robotics endeavors. The arm is built with standard *EasyRobotics* bricks, object detection is achieved using a Kinect sensor and both elements are integrated with a *TurtleBot* platform.

This prototype is capable of detecting and grasping small soft objects, like light clothes. It has trouble with larger or rigid objects, since only a basic centroid grasping strategy has been implemented. Future work could address more elaborated grasping strategies and the possibility of adding one rotating degree of freedom to the pincer. Other limitations are due to the Kinect sensor, which has a rather large minimum range, and the short reach of the arm. Despite these limitations, this project has proven to be successful at its goals and an interesting and attractive project for robotics students. Besides measuring the performance of the different modules, we have validated the correct behavior of the prototype at different environments. The demo has been running for several hours at different academic and industrial engineering venues[3], where it attracted the public and garnered positive feedback.

[3]http://www.feriazaragoza.es/matic.aspx

ACKNOWLEDGMENTS

The authors would like to thank all the colleagues in the RoPeRt group that have helped during the development and testing of this project, especially to Luis Riazuelo.

REFERENCES

[1] Riazollah Firoozian. *Servo motors and industrial control theory*. Springer, 2009.

[2] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[3] Willow Garage. TurtleBot. URL http://turtlebot.com.

[4] R. Guido, S. Fitzgerald, and D. Cuartielles. Arduino Uno, November 2011. URL http://arduino.cc/en/Main/arduinoBoardUno.

[5] J. Jones. Robots at the tipping point: the road to iRobot Roomba. *IEEE Robotics & Automation Magazine*, 13: 76–78, 2006.

[6] A Kasiński. Trajectory planning and computer analysis of kinematic problems for manipulators. *Systems Analysis Modelling Simulation*, 8(9):715–719, 1991.

[7] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for rgb-d cameras. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, May 2013.

[8] M. Krainin, B. Curless, and D. Fox. Autonomous generation of complete 3d object models using next best view manipulation planning. In *IEEE Int. Conf. on Robotics and Automation*, pages 5031–5037, 2011.

[9] Y. Nakamura and H. Hanafusa. Inverse kinematic solutions with singularity robustness for robot manipulator control. *ASME Trans., Journal of Dynamic Systems, Measurement, and Control*, 108:163–171, 1986.

[10] 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*, 2009.

[11] A. Ramey, V. Gonzalez-Pacheco, and M.A. Salichs. Integration of a low-cost rgb-d sensor in a social robot for gesture recognition. In *6th ACM/IEEE Int. Conf. on Human-Robot Interaction (HRI)*, pages 229–230, 2011.

[12] R. B. Rusu and S. Cousins. 3D is here: Point cloud library (PCL). In *IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 1–4, 2011.

[13] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Computer Vision Pattern Recognition Conference*, 2011.

[14] Eben Upton and Gareth Halfacree. *Meet the Raspberry Pi*. Wiley, 2012.

[15] A. Viklund. EasyRobotics, 2006. URL http://www.easyrobotics.fr.

[16] Mohd Ashiq Kamaril Yusoff, Reza Ezuan Samin, and Babul Salam Kader Ibrahim. Wireless mobile robotic arm. *Procedia Engineering*, 41:1072–1078, 2012.