Home Security Prototype Device

Sean McSheehy
University of Massachusetts Lowell
sean_mcsheehy@yahoo.com

Erik Cowley
University of Massachusetts Lowell
ecowley@cs.uml.edu

ABSTRACT
In this paper we solve the problem, How can a robot detect if an intruder is trying to surrender? And if they arent, should we shoot them?. This paper discusses how we detect would-be intruders using the Bilibot. We later discuss the modifications we made to the robot which iclude attaching the Nerf dart gun to the arm. We also discuss the analysis of our results and what we learned. Finally, we present ideas for future work in this field.

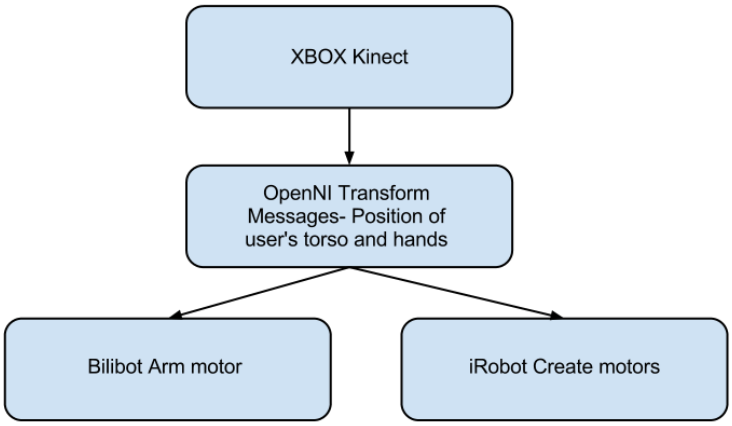
Author Keywords
Bilibot, XBOX Kinect, iRobot Create, openni, skeletal tracking,

INTRODUCTION
Our idea for our final project was to create a Home Security Prototype Device using the Bilibot and a Nerf dart gun. We use the XBOX Kinect, that is mounted to the top of the Bilibot, to detect any intruders. The base of the Bilibot is a iRobot Create that will rotate towards the intruder. The Bilibot is equipped with one degree of freedom arm that will aim/fire the Nerf dart gun using the Bilibot arm and claw.

Skeletal tracking has traditionally been used to provide a means of natural interaction with machines. Our project takes advantage of this by interpreting natural gestures that an ”intruder” might make, to determine whether that person is a threat (LaViola). Our hardware, the XBox Kinect, has been used extensively for similar purposes; however, it is not able to detect persons from as far away or as precisely as a human opponent might be able to. (Pheatt)

PROJECT DESCRIPTION

An overview of the tracking and movement of the Bilibot:



OpenNI Tracking
Our first priority to get this project to work was to track any would-be intruders. After much research, we found that we would be able to use the openni_tacker stack that is bundled with ROS. While running roscore in a separate window, we opened a new terminal window and entered the command rosrun openni_tracker openni_tracker. The allowed us to test if people were being tracked. Once we were able to determine that users were being tracked, we need to extrapolate the skeletal data. We first tried this using the command rosrun tf tf_echo, which should have produced the transform data for each limb of each user that was being tracked. We discovered the the skeletal data was only available after the user assumes the Psi pose. Once calibrated using the Psi pose, the skeletal transform data was readily available to us. For this project, we assume that each person enters the area after assuming the Psi pose.

Once we were sure that we were tracking the user, we needed the Bilibot to react appropriately. We decided that we wanted the Bilibot to rotate appropriately to where the user is. We did this by using the /torso transform from the skeletal data and publishing to the /Twist topic with the appropriate data. We tested this by opening a new terminal window and issuing the command rosrun openni_tracker openni_tracker. This would set the angular rotation along the z-axis of the Bilibot via the attached iRobot Create that serves as the base of the robot.

Bilibot Arm Movement

We next decided to implement the Bilibots arm that would house the Nerf dart gun. Our idea was to have the arm extend when an intruder was detected and then retract if the user had surrendered. We tested the movement of the arm by issuing the command rosservice call /set_arm_pos 100. This command would lower the arm approximately half way. The acceptable values are between 0 - 255, where 0 is fully extended and 255 is fully retracted. After a multitude of hours spent testing different combinations and variations of arm poses, we decided that we would use the transform from each hand to the torso to determine if the intruder was surrendering. If the user had both hands above their head, the arm would retract and not fire any darts. Once we were confident that the user was being tracked and the robot was acting appropriately, we implemented some further improvements.

Accounting for Depth

We took into consideration that the user might be at different distances and adjusted the robots aiming logic. The robot will still rotate to keep the intruder directly in front of it. Depending on whether the robot detected that the user was further or closer away, the angle of the arm would either raised or lowered to get the appropriate trajectory. We were able to get the robot claw working with the command "rosservice call /toggle_hand_state." The claw would somehow fire the gun after giving the intruder a warning to raise its hands above its head. If the user tried to mess with the robot by making it repeatedly raising and lowering its robot arm, the robot fire a shot into the intruders knee cap!

ANALYSIS OF RESULTS

Inputs and Outputs

The inputs of the XBOX Kinect would be the pose of people being tracked and the output would be the skeletal data. The input for the arm/claw was be the skeletal data of the people being tracked. The output for the arm/claw is for a Nerf dart to be launched at the person being tracked. Correctness will be determined by the enemy being hit with a Nerf bullet.

Testing

We tested this prototype at different stages for numerous hours inside room 302 in Olsen Hall located on North Campus at UMass Lowell. Each of us took turns standing in front of the Bilibot trying to figure out the openni_tracker. Success was eventually achieved as we observed the the Bilibot rotating its base to track the intruders and later when the Bilibot arm was raised and lowered appropriately towards the intruder.

Success was determined when we demonstrated our project for faculty and students of the Computer Science Department on May 3 , 2012. We were repeatedly and successfully ran our code on the Bilibot and got the expected results. The users was tracked. The robot rotated towards their position while the arm lowered and the the claw moved, indicating that the gun would be fired. The arm also retracted when the user raised both hands above their head, as expected. Because we did not attached the Nerf Dart gun to the arm, no one was harmed during our demonstration. We received much on this day and we were happy to be a part of it.

DISCUSSION

To get all of these desired actions to perform properly all at once, we needed to have a number of commands issued in a certain order. Some commands needed to be executed more than once. Our code was written to file called move_arm.py that was located in a folder in the home directory. The order for successful operation of this project are as follows...

Open new terminal window and enter "roslaunch bilibot_bringup minimal.launch"
Open another new terminal window and enter "rosservice call toggle_create_power"
Open another new terminal window and enter "roslaunch bilibot_bringup robot.launch"
Open another new terminal window and enter "rosrun openni_tracker openni_tracker"
Open another new terminal window, change to the appropriate folder and type "python move_arm.py"

We discovered that we were unable to use RVIZ while running the rest of our project. We also learned that to use the openni_tracker, users need to be calibrated in order to get the transform information. This is achieved by having the user assume the Psi pose. The practice of having users assume a specific pose to calibrate tracking, though not ideal for the purpose of a defense system, is fairly standard of contemporary work. It is a considerably less costly way of finding new users than the alternative, of identifying users in arbitrary positions (Suay).

OpenNI Frames and Transforms

Openni_tracker operates by publishing the transforms between two frames. Frames may be attached to the Kinect itself (the absolute frame) or they might represent the location of a body part of a tracked user. The transforms are ROS tf transforms, and represent the linear and angular difference in position between two frames. We tracked the position of intruders by taking the transform of the user's torso against the absolute frame. The position of the hands were determined by taking the transform of the user's hands against the torso.

We had several issues working with frame transforms, most of which were eventually resolved.

Firstly, frames tend to sporadically fall in and out of existence. We believe this is because openni_tracker sometimes loses track of single members of users, without losing track of the user as a whole. This issue was resolved by adding checks to make sure frames exist before each time they are used.

Secondly, transforms are not always available at the time that the program needs them. This might have been because openni_tracker is attempting to track a large number of targets at once, and it might not have current, complete

information on all of them at any one time. This issue was fixed by telling the transform listener to wait for transforms to become available before looking them up.

Various other miscellaneous errors were encountered throughout testing; however, we found that the tracking worked "good enough" after the two issues above were solved, and we decided to shift our focus elsewhere.

Arm Control

Since the Bilibot offers no information regarding the arm's actual position, determining where the Bilibot was aiming at any given time turned out to be a rather difficult problem. This was an important issue since the bilibot needs to know where it's aiming so it can shoot when it is on target.

Observing that the arm always moves at approximately the same speed, we implemented a position estimator by using the last known position and the time since last update:

$$p' = p + \omega t$$

where p' is the new position, p is the previous known position, ω is the constant angular velocity of the arm, and t is time since last update.

Tracking Multiple Targets

At an early stage, we found that calibrating the Kinect was tedious because there was no good way of determining which user numbers were currently being tracked. For example, our program might be set to respond to "user 1" but for some reason, openni_tracker defines the user it is tracking as "user 3". Because of this, we decided to have the program cycle through users rather than track a static user number, and whenever it comes across a user number that it is tracking, the Bilibot performs its routine on that user.

This solution lead to difficulty when openni_tracker found multiple users; it would become confused and jiggle back and forth between each user, unable to decide which one to focus on. The solution to this was that the program saved the user number of the first user it found, and from then on would only focus on that user. If openni_tracker looses track of that user, or if the Bilibot "shoots" that user (in which case the program assumes that user to be dead), it will begin to search for a new target.

Final Results

During the demonstration of our project, we noticed some behaviors that did not come to our attention during our research. Because there were multiple people in the background, sometimes it was difficult to start calibrating the user. We also noticed that it was much easier to track users that wore solid primary colored clothing such as a red sweatshirt. Users that wore gray colored clothing were more difficult to begin tracking. Sometimes when users slowly raised their arms to assume the "surrender" pose, the expected actions were not triggered in the Bilibot. We realized that we actually required a very specific hand position for the Bilibot to recognize a surrender, and unfamiliar users sometimes had difficulty assuming this position. In the future, we will have less rigidity in our requirements for natural gestures. More research on this project would be needed to come to a more specific diagnosis.

The multi-user tracking did not work as well as we had hoped; unfortunately, we did not have a chance to test the tracking on crowd prior to the exhibition. We had a problem where the Bilibot would attempt to continue tracking a user after he or she had disappeared, which forced us to restart the program between demonstrations. We believe that this is because openni_tracker continues to "remember" targets after it can no longer see them (Suay); more research would have to be done to figure out a good solution to this issue.

CONCLUSION

For future work, other poses would generate different actions from the Bilibot. Other future development could include having the Kinect detect speech for disarming or echo warnings via the built in microphone and speakers.

ACKNOWLEDGMENTS

The work described in this paper was conducted as part of a Spring 2012 robotics course, taught in the Computer Science department of the University of Massachusetts Lowell by Prof. Fred Martin. We would like to acknowledge all of our classmates who may have helped with this project or otherwise inspired us.

REFERENCES

[1] ROS.org http://www.ros.org/wiki/

[2] openni_tracker http://ros.org/wiki/openni_tracker

[3] Suay, Halit, and Sonia Chernova. "Humanoid Robot Control using Depth Camera." *Proceedings of the 6th international conference on Human-robot interaction.* (2011)

[4] Pheatt, Chuck, and Jeremiah McMullen. "Programming for the XBox Kinect Sensor." *Journal of Computing Sciences in Colleges.* 27.5 (2012)

[5] LaViola, Joseph, and Daniel Keefe. "3D spatial interaction: applications for art, design, and science." *SIGGRAPH '11 ACM SIGGRAPH 2011 Courses.* (2011)