Reducing Cumulative Odometry Error with Overhead Camera Localization in the Soar Cognitive Architecture

Minxing Pan and Nicholas C. Matton
The University of Michigan

Abstract, todo

Keywords: robot mapping, localization, linear algebra, computer vision

Introduction

Intelligent robots need a navigation map of their surroundings in order to efficiently travel and complete tasks in their local environment. In simulations, the exact location of a robot is usually consistent with that of the robot's own navigation maps. Physical robots, however, can introduce odometry error into the mapping process. Odometry is calculating relative position based on the measurements of sensorimotor data. While odometry provides easily accessible real-time robot pose, its accuracy is dependent on if the wheel revolutions measured by the encoders can be fully translated into linear displacement relative to the ground (Borenstein, 1996).

For our project, Anki's Cozmo robots are used to present a method of correcting odometry error in mobile robots. Cozmo is a fist-sized robot capable of exploring unknown environments and interacting with cubes. It builds a two-dimensional navigation memory map (nav map) of its surroundings as it travels. The nav map is built mostly based on where Cozmo's camera picks up the objects and where Cozmo detects drops and visible edges. Unlike standard occupancy maps, which only compute the probability of a grid space being occupied by an object, the nav map also annotates the type of object in the occupancy grid. The nav map is stored as a quad-tree, in which leaf nodes hold the contents such as cubes and chargers, and all other nodes split into four children. Child node indices are stored in the x, y orientation shown in the figure (Anki, 2016).

In addition to the nav map, Cozmo relies on odometry to estimate its position and rotation (Cozmo pose) as it travels. Cozmo uses a right-handed coordinate system, in which the y-axis is to the left of the x-axis, and counterclockwise rotation is considered positive. When first initialized, Cozmo recognizes its current pose as the Cozmo origin, such that its x, y, z, rotation, pitch, and yaw are all zeros, and the direction it faces towards is the positive x axis.

In Cozmo odometry, the perceived travel distance is based on the encoder values of the motors, and the actual travel distance is the displacement of Cozmo relative its previous position. Some of the systematic odometry errors include "unequal wheel diameters" and the "misalignment of wheels" (Borenstein, 1996). Because Cozmo drives with treads, the horizontal instability of the treads while driving contributes to the misalignment. The wear and tear of tread on either side causes unequal diameters. Consequently, the Cozmo odometry can be inaccurate. For example, when Cozmo turns, Cozmo's perceived sensorimotor data can be a few degrees different from the actual travel data. This cumulation in odometry drift causes the Cozmo origin to deviate, leading to an inconsistent perceived robot pose with regards to the actual robot pose. The cumulative odometry error causes problems in robot exploration and navigation over time.

Our project runs programs controlling the Cozmo robots on the Soar Cognitive Architecture. "Soar is a general cognitive architecture that integrates knowledge-intensive reasoning, reactive execution, hierarchical reasoning, planning, and learning from experience, with the goal of creating a general computational system that has the same cognitive abilities as humans" (Laird, 2012). Our program uses the Cozmo-Soar Interface to interact with the Cozmo robots. The Cozmo-Soar Interface is a python-based interface between Soar and Cozmo based on the Soar Markup Language (SML), PySoarLib, and the Cozmo SDK (Minniger, 2018). "The purpose of the interface is to allow a Soar agent to control a fully-embodied Cozmo robot, thereby embodying the Soar agent in the real world and enabling cognitive experiments to be run on a small, flexible, and robust platform" (Boggs, 2019).

Method

We are approaching the problem of inconsistent perceived robot pose and the actual robot pose with an overhead camera system that uses localization to correct the Cozmo's pose. The vision processing is done using OpenCV version 4.1.1. OpenCV is an open-source computer vision library. Other researchers have modeled odometry error using Simultaneous Localization and Mapping (SLAM) with advanced sonar systems (Kleeman, 2003). Another researcher implemented scale-invariant visual landmarks to correct local odometry while only using odometry to predict the region to search for landmarks (Se et al., 2001). Both methods are effective but

require more computing power, expensive equipment, longer processing time, and more complicated math models. Our method of overhead camera localization requires less computing power and inexpensive equipment, but can achieve similarly accurate results.

Camera Calibration

The Logitech HD Pro Webcam C920, Widescreen Video Calling and Recording, 1080p Camera, Desktop or Laptop Webcam is selected for our overhead camera system. Camera Calibration is necessary because each camera may introduce some distortion to the images it perceives. The two major types of distortion are radial distortion, which causes straight lines to appear curved, and tangential distortion, which causes some areas in the image to look nearer than actual (Bradski, 2000).

The OpenCV library provides a function to calculate the distortion coefficients needed to correct the images with a calibration process using a checkerboard image. In this project, a nine by six monochromatic checkerboard image provided by OpenCV was used to calibrate the camera. Our program collects a stack of 100 different images of the checkerboard at different angles and distances to the camera. Then our program uses the OpenCV function of calibrating the camera to calculate the distortion coefficients. After calibrating the camera once in the lab environment with ambient lighting, the calibration data can be used to undistort new images streamed by the camera prior to any further image processing.

Pose Estimation

Since cumulative odometry drift causes the Cozmo origin to deviate, there needs to be a reliable method to estimate the robot pose and recalibrate the Cozmo origin. Pose estimation can be done based on finding correspondences between objects in the real world and their two-dimensional image taken by the camera (Bradski, 2000). While one could estimate robot pose with a multi-objective Convolutional Neural Network, that process requires hundreds of robot images for training (Miseikis, 2018).

In this project, binary square fiducial markers are selected, because an individual marker's four corners provide enough correspondences to establish a pose relative to the camera. The Cozmo robot can be identified by an ArUco marker placed directly on top of it. An ArUco marker is a square marker with a distinguishable black border that allows for fast detection in an image and an inner binary matrix that encodes an identifier (Id) (Garrido-Jurado et al., 2014). Because only two to three identifying markers are needed, the smallest ArUco dictionary are selected for the fastest processing time. ArUco markers that are four bits by four bits markers with 200 pixels in each and one bit in the border are printed and cut out. One marker is taped directly on top of Cozmo while the camera is placed straight above the Cozmo exploration environment.

Our program can then estimate the robot pose with images streamed from the calibrated camera. The OpenCV algorithm can estimate the pose based on a single ArUco marker and the camera calibration data. For every image frame fed to the program, the function returns the rotation and translation vectors of each detected marker in the image. The two vectors transform a three-dimensional point of an object to a frame relative to the camera (Bradski, 2000). The pose obtained from this method is of the marker in relation to the camera. Therefore, additional processing is needed to turn the pose into a right-handed world coordinate system that can be easily translated to and from the Cozmo coordinate system.

Another ArUco marker is then placed at the bottom left corner of the Cozmo exploration environment and set that as the origin of our world coordinates. Both the translation and rotation vector of the world origin are then subtracted from those of the robot pose marker. This way, the robot pose is no longer moving relative to the camera, but rather in relation to a fixed point in the Cozmo exploration environment. Furthermore, because the camera captures a mirrored image of the actual setup, the y-axis and the yaw need to be flipped to make the image consistent with the real world.

After the calculation of the translation and rotation vectors of the robot pose in relation to the world origin in a right-handed world coordinate system, the two vectors needs to be translated into x, y, z, roll, pitch, and yaw values. Conveniently, the translation vector's three values are the x, y, and z values of the robot position coordinates that can be used without any other calculation. The rotation vector, however, requires additional computation to transform into euler angles (roll, pitch, and yaw). First, Rodrigues' Rotation Formula converts the rotation vector to a three by three rotation matrix. Then, an implementation similar to that of MATLAB is modified to convert the rotation matrix to euler angles (Mallick, 2016). Last but not least, the only rotation needed for our application is that with regards to the z-axis (yaw), so only the third value of the Euler angles is needed.

Linear Transformations

A method to acquire the real time robot pose in the world coordinate system has been established so far. Cozmo, however, has its own coordinate systems described in the Introduction section. Our program can convert to and from the two coordinate systems.

First we will explain some common Linear algebra terms. [Linear algebra definitions and how they relate to our system here] [explain how a coordinate system is a basis with the axes' being its members and the plane that the system is on is a vector space]

There are two differences in the coordinate systems: rotation and translation. A rotational difference creates an inconsistency in bases between the two coordinate systems. As shown in Figure 420, the same vector represented by the coordinates [1/sqrt(2), 1/sqrt(2)] in one coordinate system is represented by the coordinates [0, 1] in the other. This is because the rotation caused the systems to have different bases. To convert between the different bases, calculate the change of basis matrices for these two coordinate systems. Then use these matrices to transform a vector in one basis to the other basis by multiplying that vector by the change of basis matrix. Because the axes' for both coordinate systems are orthogonal, this implies that both coordinate systems have orthogonal bases. This means simple rotation matrices are needed to convert between the two systems. These matrices will be equivalent to our change of basis matrices. Therefore, to create our first change of basis matrix, find the difference in rotation between the robot origin and the world origin and feed that angle into the rotation matrix (see Figure 69). Then create the other change of basis matrix by taking the inverse of the first change of basis matrix.

The second inconsistency between the two coordinate systems is translation. The Cozmo origin is not at the same coordinates as the world origin, so a vector must also be translated so that it is relative to the correct origin. This can be achieved by keeping track of the Cozmo origin's position relative to the world origin in an origin difference vector (ODV). Figure 42069 demonstrates why the origin difference vector is used to convert between the two coordinate systems' origins. To find the origin difference vector, first use the camera to find the vector that represents cozmo's pose relative to the world origin (CWO). Then access the Cozmo robot to find the vector that represents cozmo's pose relative to the cozmo origin, and use the change of basis matrix to convert that vector to world coordinates (CCO). This conversion must be made so that the vectors are relative to the same basis. Then, lastly, subtract CCO from CWO. This will return the position of cozmo's origin in world coordinates, or the origin difference vector. You can see visually how this would work in figure 33.

The change of basis matrices and the origin difference vector can easily be used to transform coordinates with a single matrix multiplication and a single vector subtraction or addition (depending on the direction of the translation). This calculation can be seen in figure 123. When changing a given vector from world coordinates to cozmo coordinates, subtract the origin difference vector from the given vector. This will return the given vector relative to the cozmo origin, but in the world's basis. By then multiplying by the correct change of basis matrix, our program will return the given vector relative to cozmo's origin and cozmo's basis. When changing a given vector from cozmo pose to world pose, first multiply the given vector by the correct change of basis matrix. This will return the given vector relative to the cozmo origin in the world's basis. By then adding the origin difference vector, our program returns the given vector relative to the world's origin and the world's basis.

Update Robot Pose

Our code encapsulates the said functionality in a Camera Localizer class in Python. To not slow down the overall Soar and Cozmo program, the Camera Localizer spuns a separate thread. The update transformation and conversion functions are periodically called in the Rosie branch of CozmoSoar to calibrate robot pose.

Experiment

Results

Discussion

fuck this worked!!!!(Einstein, 1905)

References

Einstein, A. (1905). Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]. *Annalen der Physik*, *322*(10), 891–921. doi: http://dx.doi.org/10.1002/andp.19053221004