MAKI Oculomotor Algorithms

Abstract
The goal of this project was to develop a system by which the MAKI 3D-printed robot could track a brightly colored object using both its eyes and head as naturally as possible.

Before the project started, a version of the MAKI robot had already been assembled, and a low-level driver had been written for its Arbotix-M board, which managed the robot’s Dynamixel motors. One of the initial goals of the project was to “ROS-ify” MAKI by writing Python “nodes” compatible with the Robot Operating System, which is the accepted industry standard for interfacing with robots. In addition, a Universal Robot Descriptor File (URDF) was written that specified the various joints and links to allow for a universal interface. Building on top of these foundations, a “Smooth Pursuit” algorithm was implemented for natural eye-tracking of a brightly colored object, and a node was created that, when active, would fix eye-gaze on an object and adjust eye-position when the head was moved in order to maintain contact.

The final deliverable for the project was an assembled MAKI, which included an additional degree of freedom (in the torso) and improved internal structure. This process was documented so that future robotics students can also build MAKIs with the custom improvements.

In the future, this work will be incorporated into the collaborative Signing Creatures Project, in which researchers from Gallaudet University, Yale, USC, and IIT are seeking to use an embodied social robot and virtual human avatar pair to facilitate a hearing-impaired infant’s acquisition of sign language.

Motivation
The MAKI robot was initially designed by Hello Robo as a low-cost research platform for working with children with autism. This has since been expanded by researchers at Gallaudet University, Yale, USC, and IIT into the Signing Creatures project, which works with hearing-impaired children.

One of the primary components of language acquisition is “joint attention,” a behavior in which two individuals focus on the same object. This is significant for children learning about the world—adults are able to impart information about a particular object and children learn to associate the conceptual with the physical. For infants, this exchange often takes the form of the adult speaking the name of the object.

The motivation behind this project was to prepare the MAKI robot to be used in language acquisition studies, for which the ability to participate in joint attention is a fundamental prerequisite. The end result of this project will be the added functionality of tracking an object through combined movements of both the eyes and head, with an emphasis on making MAKI’s transition lifelike.

Technical Considerations
MAKI is an open-source 3D printed robot. The 3D printers available at Yale in the CEID are not strong enough to print using nylon material, so we ordered parts from a third party company (Shapeways) using preexisting STL files. The MAKI was assembled with several Dynamixel motors for each joint, all connected to a low level Arbotix-M board, which is Arduino-based. The MAKI servo driver had already been developed. The Arduino board connected to an Ubuntu 14.04 machine, on which we programmed oculomotor algorithms in Python. ROS (Robot Operating system) Jade was selected in order to leverage its transformation frame capabilities, message-passing system, drivers, libraries, and package-management. ROS has become the de facto standard for Robotics research; there are many user-contributed packages available.
The oculomotor algorithms used were researched and implemented by Maikel Linke for the robot Flobi, a robot with some significant mechanical differences from MAKI. Notably:

- Maki’s eye pan and eyelids are coupled, rather than functioning independently.
- Flobi has a camera in each eye (stereo vision), whereas MAKI only has a camera in its mouth. This is particularly relevant as the pairing of eye and head movement affects how visual images must be processed as their relative positions change.

**Oculomotor Algorithms**

“Smooth Pursuit” is the act of the eyes following a moving object. The main variable in this scenario is the velocity of the eye-movement, as the velocity of the object itself is subject to change and an incorrect velocity can result in a position error (either an over- or undershoot). Based on the research of Maikel Linke, a position-based algorithm is preferable to a velocity-prediction algorithm. The latter is significantly more computationally complex but does not offer a corresponding increase in accuracy. One goal of my work was to program the robot to switch from just a shift in eyes to a joint head and eye movement in order to appear as natural as possible.

The “Vestibulo-Ocular Reflex” shifts the position of the eyes such that the focal point remains the same when the head moves from its original position. A clear example of this effect is when someone shakes their head “no” yet keeps their gaze fixed on a given object (such as another person). The equations for calculating the appropriate movement are delineated in Linke’s work, though for use with Flobi.

**Resources**

Two primary lab-specific resources existed at the beginning of the project; a pre-built MAKI 1.4, and the Arbotix driver Kate Tsui had developed over the summer. The MAKI 1.4 differed from the version we built as part of the project in both internal head structure and number of joints (servos). The newer version has an additional degree of freedom in its torso, which can now pan laterally independently from the head. The driver served as a processor for communication between the computer’s serial port and the servos themselves, and was written to take commands of the following forms:

**Feedback:** \( F[MX|MN|PP|PS|PT|PL|ER]Z \)

- \( FMXZ \) // get FEEDBACK of all servo MAXIMUM POSITION
- \( FMNZ \) // get FEEDBACK of all servo MINIMUM POSITION
- \( FPPZ \) // get FEEDBACK of all servo PRESENT POSITION
- \( FPSZ \) // get FEEDBACK of all servo PRESENT SPEED
- \( FPTZ \) // get FEEDBACK of all servo PRESENT TEMPERATURE (Celsius)
- \( FPLZ \) // get FEEDBACK of all servo PRESENT LOAD
- \( FERZ \) // get FEEDBACK of all servo ERROR (AX_ALARM_LED)
- \( FDPZ \) // get FEEDBACK of all servo DEFAULT POSITION
- \( FDSZ \) // get FEEDBACK of all servo DEFAULT SPEED

**Command:** \( mm[GP|GS]xxx[IPT]yyyyyZ \)

- \( mm = \{ LR, LL, EP, ET, HT, HP \} \) means eyelid right, eyelid left, eye pan, eye tilt, head tilt, and head pan, respectively. LR has been deprecated; LL moves both eyelids
- \( xxx = [0, 1023] \), and \( yyyyy \) in milliseconds

In addition, the nature of ROS and its open-source community provided a number of other resources. The tutorials for creating a “catkin workspace”, creating ROS nodes, and publishing and subscribing to topics were particularly helpful.
## Deliverables

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Complete</th>
<th>Incomplete</th>
<th>Comments</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updated Arbotix Driver</td>
<td>*added default position and default speed functionality</td>
<td>*NA</td>
<td>*IPT is bugged in both old and new versions</td>
<td>*Updated Driver</td>
</tr>
<tr>
<td>Assembled MAKI</td>
<td>*3D-printed pieces assembled, servos attached</td>
<td>*Wiring at base of robot * nose (new design needed for updated head pieces)</td>
<td>*used heated inserts instead of self-tapping screws for ease of dis- and reassembly *new design files needed for Neck Cover Front (USC Refresh), updated neck pieces, and TBM1, which needs a symmetrical counterpart</td>
<td>*Assembly photos</td>
</tr>
<tr>
<td>ROS Interface</td>
<td>* ROS node subscribing to maki-command and publishing to various feedback topics</td>
<td>* NA</td>
<td>* Publishing current publishing position after each position is currently commented out for the sake of processing speed</td>
<td>*ROS Interface Code</td>
</tr>
<tr>
<td>URDF</td>
<td>* URDF for joints and linkages * ROS node to interface between RVIZ (simulator) and ROS Interface</td>
<td>* kinematic model of physics</td>
<td>* Order of links in file is based on hierarchy of connections, not servo IDs, so URDF Interface account for that</td>
<td>*URDF File Code * Interface * Demo Video</td>
</tr>
<tr>
<td>Smooth Pursuit</td>
<td>* ROS node that subscribes to webcam feed and applies Smooth Pursuit algorithm to determine position commands for MAKI</td>
<td>* NA</td>
<td>*Two other algorithms represented (but not active) in the file *Could add interface for selecting color to track,</td>
<td>*Smooth Pursuit Code * Demo Video</td>
</tr>
<tr>
<td>VOR</td>
<td>*ROS node that implements MAKI’s version of VOR, which latches on to an axis and maintains eye-contact when head moves</td>
<td>* Service to turn reflex on or off</td>
<td>* Need to find a new name, as VOR actually refers to constant adjustments based on unintentional movements of entire MAKI, rather than deliberate actions such as head-shaking</td>
<td>*VOR Code * Demo Video</td>
</tr>
</tbody>
</table>

## Major Difficulties

One primary issue I encountered was the learning curve; despite brief exposure to ROS Groovy a few years ago, relearning how the pieces worked together contributed to a number of inefficient coding sessions that, were I to redo them, would take much less time. This was reflected in the fact that the tasks I worked on towards the end of the semester were completed much more quickly.
Given my initial unfamiliarity with the internals of the robot, ordering pieces for the new MAKI was difficult since I didn’t have a clear vision or understanding of how the numerous components fit together (especially the new pieces). Fortunately, Kate was able to step in and ensure that the correct parts were ordered.

Finally, I ran into issues with flaws in the provided STL files. Some of them didn’t seem to be designed with assembly in mind, as they required post-production modification (with Dremel, Drill, and Hacksaw) in order to fit together. This made the assembly of the new MAKI substantially more difficult.

Reflections

Over the course of the project, I gained a number of valuable skills. First and foremost, I gained an understanding of ROS, which is broadly applicable in the realm of robotics. I also used Github’s services for the first time, having previously used other Version Control websites. Finally, I used a number of hand tools for the first time, including a Dremel, Drill, and Hacksaw.

In terms of meeting my proposed deliverables, I think I successfully completed the core functionality for each component, with the exception of VOR. The Kinematic aspect of the URDF was not necessary for the stated goals, but could be useful going forward. I also would have liked to add a color-selector for Smooth Pursuit, as hard-coding values is never ideal. Finally, a service for turning the VOR on and off will be important for incorporating it into broader behaviors.

Going into the project, I had two stretch goals: adding additional lifelike behaviors (e.g. rolling eyes) and an interface that would allow users to see what MAKI sees, select an object, and have MAKI turn to that position. The first would be fairly easy to complete, as all it requires is a script that would sequence commands and sleep()s to achieve the desired behavior (the only difficulty would stem from querying Present Position and generating commands off of that). The second would be a bit harder, as I haven’t done anything with mouse-clicks as of yet, but using the code in camera-listener.py I can imagine having a service that turns mouse clicks into command using the same calculations.

Models of ROS Nodes

Triangle: prewritten, Circle: node or file I created, Rhombus: ROS topic.

Arbotix Interface:

This node communicates through the serial port with the Arbotix Driver (which I updated slightly but otherwise left alone) to send commands to MAKI’s servos. It subscribes to maki_command, to which other nodes publish commands, and it publishes to various feedback topics (one for each type of feedback). They are: maki_max_feedback, maki_min_feedback, maki_position_feedback, maki_max_feedback, maki_temp_feedback, maki_speed_feedback, maki_error_feedback, maki_default_position, maki_default_speed.
URDF Model/ Interface:
RVIZ reads the URDF file and runs a simulator with a GUI that publishes to joint_states, which is subscribed to by the MAKI-URDF-Interface node (necessary to turn joint_states into commands). This is subscribed to by the Arbotix Interface depicted in the first diagram.

Smooth Pursuit:
The prewritten usb_cam node (incorrectly depicted as an oval in this diagram) publishes the image it receives from the webcam to image_raw, to which camera-listener.py subscribes. camera-listener.py applies HSV color thresholding and the Smooth Pursuit algorithm to determine which commands to send to maki_command.

VOR:
This diagram shows one potential use of VOR: tracking or finding an object with smooth pursuit, locking to it, and maintaining eye-contact using VOR, which subscribes to maki_command and, when enabled, publishes the reactionary eye-movements. As it stands, the only tested usage of the VOR is in response to one-time commands, not behaviors.

Walkthrough
Setup
This walkthrough assumes you have set up a catkin workspace (as per the ROS tutorials), have
ROS Jade installed, and have Arduino set up. Additionally, completing the ROS tutorials will be helpful
for understanding how pieces fit together. All of these processes are well-documented! Then, download
the git repository and move it into your catkin_workspace (path should end up as
catkin_ws/src/maki_oculomotor). Make sure you’ve set the source path (this is outlined in the tutorial,
but is very important!). Additionally, the MAKI-Controller-Lite should be moved into wherever your
Arduino sketchbook is setup. To make a catkin_workspace (in this case named ~/catkin_ws):

$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
$ cd ~/catkin_ws/
$ catkin_make
$ source devel/setup.bash

Communicating with MAKI
- Run roscore
- Navigate to maki_oculomotor/scripts and run ./maki_oculomotor/scripts/MAKI-Arbotix-
  Interface.py
- Publish to the topic with commands such as “rostopic pub -l --once /maki_command
  std_msgs/String HPGP500Z”

URDF/RVIZ
- Run roscore
- Run command: roslaunch maki_oculomotor makidemo.launch

Smooth Pursuit
- Run roscore
- Setup webcam
  o rosparam set usb_cam/pixel_format yuyv
  o rosparam set usb_cam/video_device “/dev/video2”
    ▶ use “ls ls /dev/video*” to find appropriate name
  o rosrunc usb_cam usb_cam_node
- run ./maki_oculomotor/scripts/MAKI-Arbotix-Interface.py
- run ./maki_oculomotor/scripts camera-listener.py

VOR
- Run roscore
- run ./maki_oculomotor/scripts/VOR
- send commands to HP and HT
  o optional: have Smooth Pursuit running in back

References


Google Drive with Past Assembly Instructions and Images:
[https://drive.google.com/folderview?id=0B9inX4_RGBh9YXNtSmdqeXFyOWM&usp=drive_web](https://drive.google.com/folderview?id=0B9inX4_RGBh9YXNtSmdqeXFyOWM&usp=drive_web)