UNIVERSITY OF TORONTO

Android Robotic Manipulator

Final Report

ECE1778 – Creative Applications for Mobile Devices

René Rail-Ip, Hao Yan, and Paul Grouchy

2013/4/12

Word Count: 2399

Table of Contents

Table of Figures

Introduction

In the robotics and mechatronics field, smart systems are required to have accurate and precise sensors in order to perceive and react to the environment around them. Many sensors have been developed that measure force, speed, pressure, magnetic field, temperature, proximity, light, sound, and more. However, technology exists that incorporates all of these sensors onto one fairly small, affordable and portable device: the smartphone. Furthermore, the smartphone contains varyingly fast processors, large touch screens for display and user input, and the ability to develop applications that can access all of these functionalities.

Lately, there has been a large increase in the performance to cost ratio of MEMS (Micro Electro Mechanical Systems) accelerometers and gyroscopes, which are used in smartphones. With this increase and the growing acceptance for consumer and household robotics, it is natural to begin to think of using this extraordinary device and its sensors for robotic applications. The applications have grown beyond the household as well; industrial robots are becoming less expensive and there is more expectation for intuitive and wireless control of robots.

The Android Robotic Manipulator (A.R.M.) project aims to provide an initial proof-of-concept that MEMS accelerometer and gyroscope technology can provide accurate and precise enough data to provide motion control of a robotic entity. The overall goal of A.R.M. is "**to allow for wireless and intuitive real-time control of a robotic manipulator with an Android smartphone.**" The robotic manipulator used for this project is a four-degree-of-freedom arm made of Robotis AX12A motors designed by CrustCrawler. The project aligns with the research of the Apper as described below.

Apper Context

René Rail-Ip is the Apper for the A.R.M. group project. He studies in the field of Electrical and Computer Engineering with a specialization in robotics and mechatronics, sensors, and hardware implementations. The goal of his MASc thesis is to develop a platform to benchmark and evaluate the performance of various MEMS accelerometers. In order to do this, accurate and repeatable motions must be applied to the accelerometers while recording the raw data output from the sensors. This requires for the accelerometer chips to be mounted onto a motion platform while outputting data to a computing unit, which can save the data.

The primary objective of this research is to evaluate accelerometers' capabilities in applications as an accurate and precise inertial measurement unit, which is an instrument that

can track the position of some moving object through an environment. This involves integrating the accelerometers' and gyroscopes' outputs in order to obtain positional data, whose accuracy and precision depend largely on the algorithms and processing power being used.

There is a particular interest in the implementation and application of sensors for robotic technologies. Smartphones today include amazing sensor technology to gain information including applied forces, applied angular speeds, local pressure and temperature, proximity information, and more. Because robots rely on accurate sensors to understand the environment they are operating in, smartphones can become a very useful and affordable tool for many robotic applications. The specific use of the accelerometers in the smartphone are of interest to the research being conducted by René and developing a smartphone app that can make use of this sensor data could pave the path for development of future applications to evaluate and benchmark the performance of accelerometers within different smartphone devices.

Throughout the project, a basic platform was set up to send motion commands to a robotic manipulator while transmitting data between devices over a wireless network. The wireless data communication was a two-way protocol with one end sending integrated accelerometer, gyroscope, and magnetometer data and the other end sending confirmation signals. Designing and building the entire system was good practice in developing an intelligent motion platform that can communicate between devices, which is what will be needed for René's research.

Finally, the A.R.M. application requires that the accelerometer and gyroscope outputs be mathematically integrated to obtain positional data, which is sent to control the robotic manipulator. The algorithms used and researched to most optimally reduce sensor noise and accurately obtain displacements from accelerations and velocities will be of great value to the research in developing an inertial measurement unit that utilizes accelerometers and gyroscopes to track humans or mobile robots within an environment. There are various different types of algorithms and techniques used to obtain accurate and precise displacements from the accelerations but the sensors suffer inherently from noise and temperature problems. The successful utilization of the accelerometer and gyroscope data to control the robotic manipulator completes a proof-of-concept for the use of these sensors as accurate displacement measuring devices.

Overall Design

Figure 1 shows the overall design of the A.R.M. system including the client and server modules.

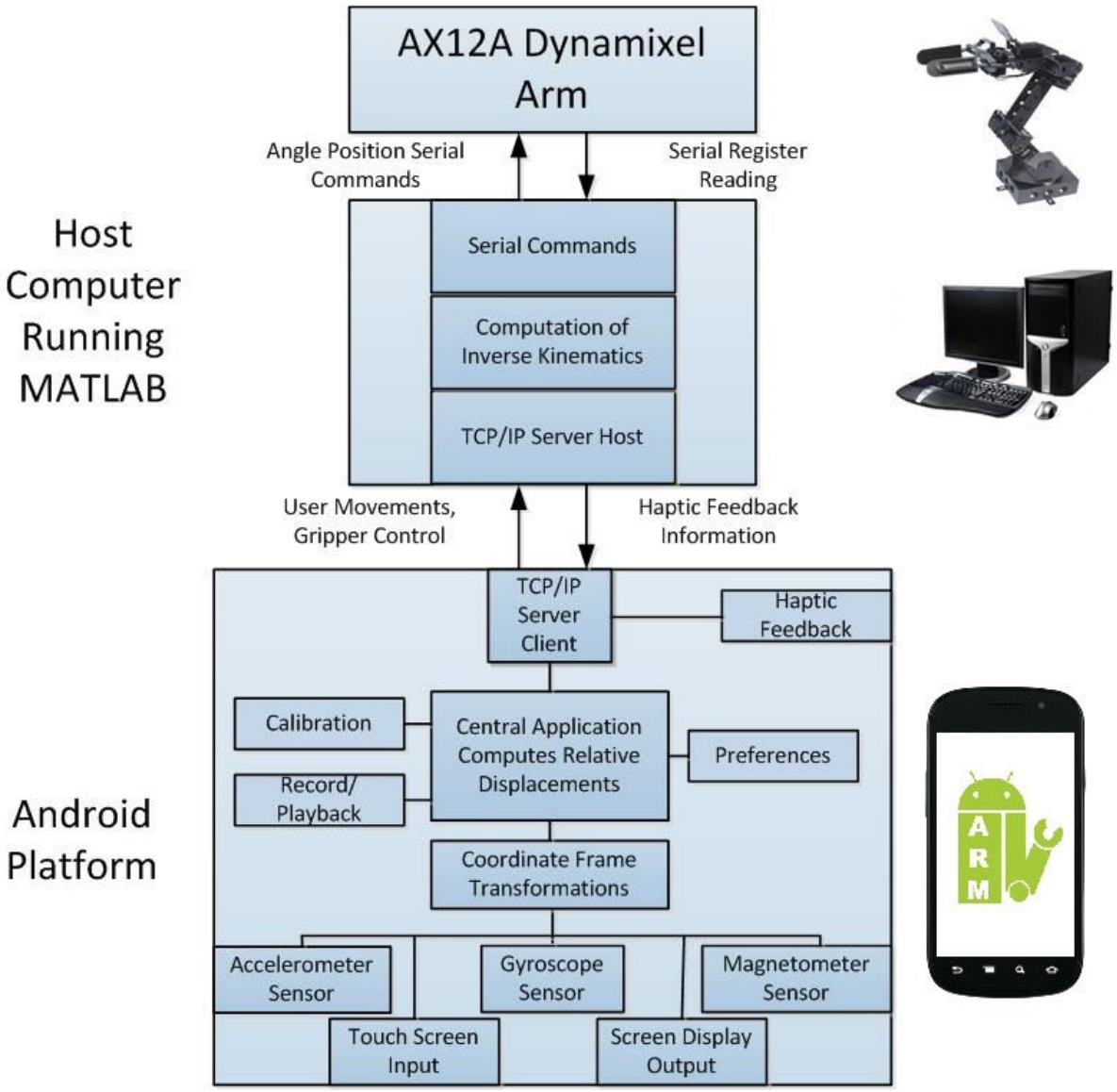

Figure 1: Block Diagram

There are three major components to the A.R.M. application: the app on the mobile device, the host computer running MATLAB and the AX12A Dynamixel robotic arm. On the mobile device side, the A.R.M. app collects the movements of the device by the end user, but only when the user is holding down the main button. The app then translates the device's movement outputs into an absolute, real-world coordinate frame using the device's

magnetometer and rotation sensors. It is at this point that various noise-reduction steps are implemented. There are three techniques used: thresholding, calibration, and speed decay. Thresholding is used to ignore small acceleration values, calibration subtracts average accelerometer values taken from when the device was calibrated at rest by the end-user, and speed decay subtracts a fixed value per second from the speed to counteract the accumulation of integration errors (device displacement is calculated by integrating accelerometer data to get speed and then integrating speed to get displacement and accelerometer errors are accumulated during these steps). Also, it is at this point that displacement values are scaled by a user-defined sensitivity value. Relative device displacement values are periodically sent via Wi-Fi to the host computer. If the user is not holding down the main button, zeros are sent. The current gripper slider bar value is also sent with this data. If an axis' movement is locked by the user or by the Wrist Mode/Arm Mode functionality, a zero is sent for that movement. If a periodic data send fails, the app assumes the Wi-Fi connection has been lost. The mobile app was set up for intuitive usability, with the majority of the user interactions happening through the main button (see screenshots below).

On the host computer side, relative device movements are received from the device over Wi-Fi and are used to compute (using inverse kinematics) the motor signals required to move the robotic arm's end effector in a manner that mimics the mobile device's movements. The custom code running in MATLAB must first detect whether the device is sending Wrist Mode or Arm Mode data by examining which data values are zeros, and then use the non-zero data to calculate the specified arm motor movements. The gripper is adjusted if the gripper slider value has changed. The MATLAB code also detects when requested movements are outside of the robotic arm's motor ranges, in which case the mobile device is notified via a signal over Wi-Fi (the device vibrates when it receives such a signal, thus implementing haptic feedback). Furthermore, movement complete confirmations are sent back to the device over Wi-Fi for use in playback functionality. The computed motor and gripper signals are sent via a wired USB-to-Serial connection to the robotic arm, which then moves accordingly.

Record functionality is implemented by saving the accumulated device movements when the user hits the record button on the main screen. Playback transmits each movement snapshot sequentially to the MATLAB server, waiting for the movement complete signal from the server between sends of movement snapshots.

"Return Robotic Arm to Home Position" and "Client Is Disconnecting From Server" signals are sent from the device to MATLAB via special values in the gripper section of the regular Wi-Fi packets.

Statement of Functionality and Screenshots from App

The A.R.M. development team achieved full functionality with three caveats. On the mobile device, movements were successfully recorded and converted into absolute coordinates. Data packets were successfully transmitted over Wi-Fi to a host machine running MATLAB, which in turn was able to move the robotic arm as the user intended. The first two caveats are here, as when the user holds the main button down for an extended period of time, the speed values accumulate error, thus producing erroneous displacement values. This was partially mitigated through various noise reduction algorithms (mentioned above) and the zeroing of speed whenever the user released the main button. However, further work on noise reduction is necessary to improve functionality and usability. Furthermore, the robotic arm does not react in real-time to device movements. While the reaction times are fast, further work (including investigating Bluetooth data transmission and alternatives to our MATLAB implementation) might help to decrease lag.

Axis locking, gripper control via a slider UI, haptic feedback, sensor calibration, and the settings activity (with calibration and settings values persistent via internal storage) were all successfully implemented. Arm and Wrist Modes were also functional, along with a “Home” button to return the robotic arm to its home position.

Finally, record/playback functionality was successfully implemented, with surprisingly good accuracy. A user is able to record snapshots of the current robotic arm position and then play these positions back sequentially. This is where the third caveat comes in, as playback has accuracy issues if the user records too few snapshots. It is postulated that this is a combination of motor noise and inverse kinematics calculations that produce different robotic arm movements to a specified position with and without intermediate steps. Implementing a feedback controller might alleviate this issue.

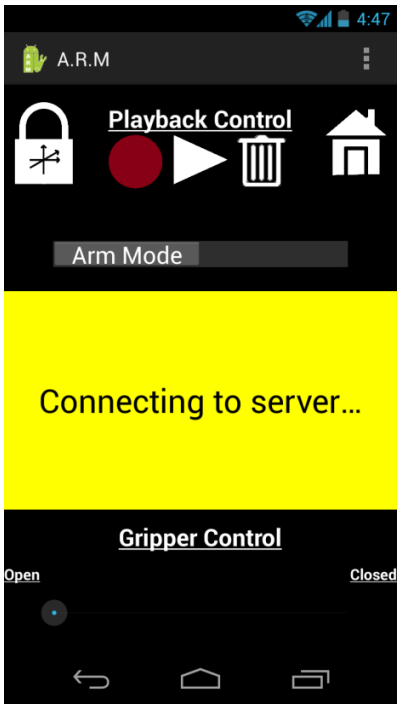

Figure 2: Connection attempt

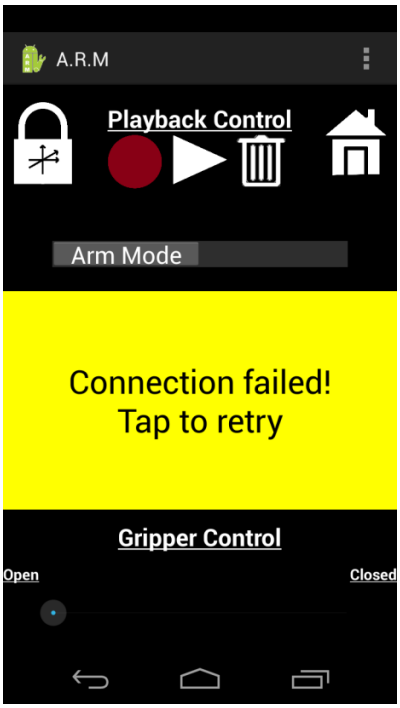

Figure 3: Connection attempt failed

Figure 4: A.R.M. connected

Figure 5: Main button pressed

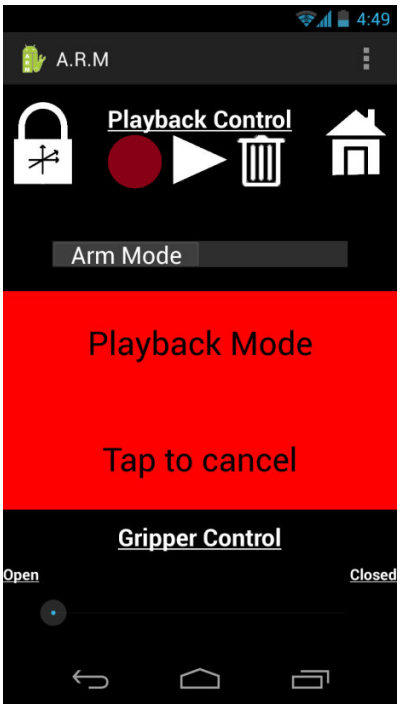

Figure 6: Playback mode

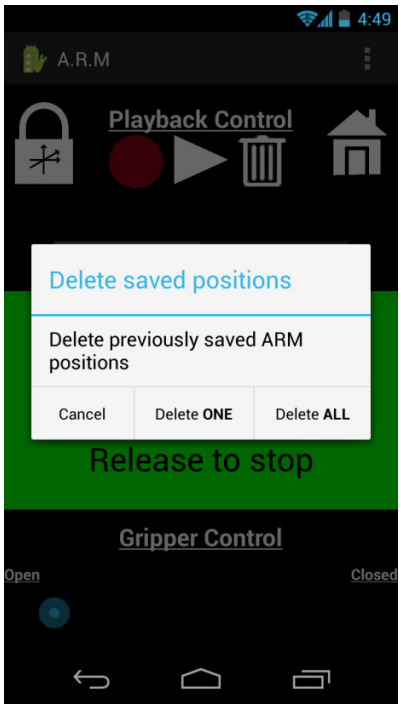

Figure 7: Trash icon pressed

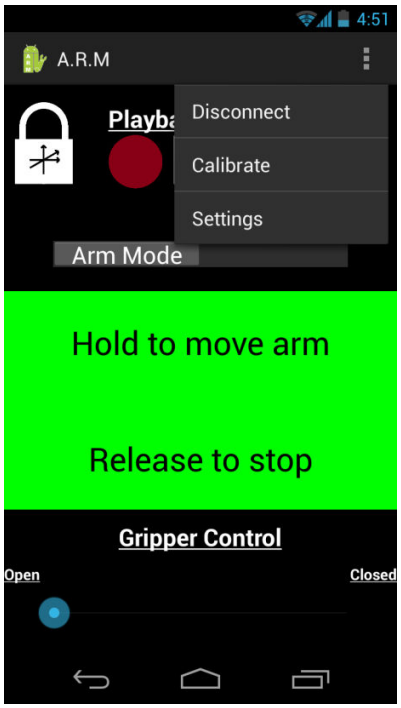

Figure 8: Settings menu

Figure 9: Calibration activity

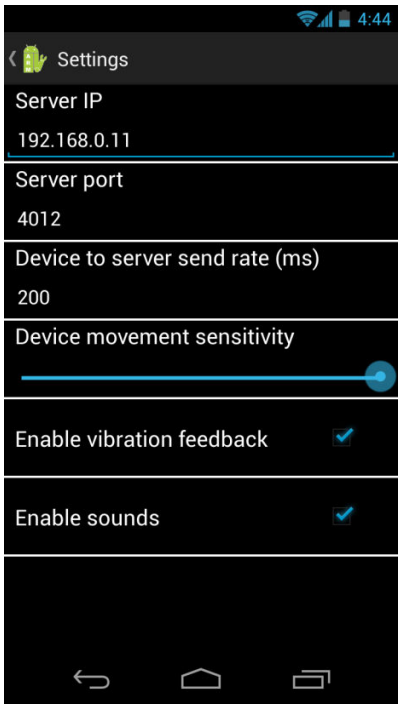

Figure 10: Settings activity

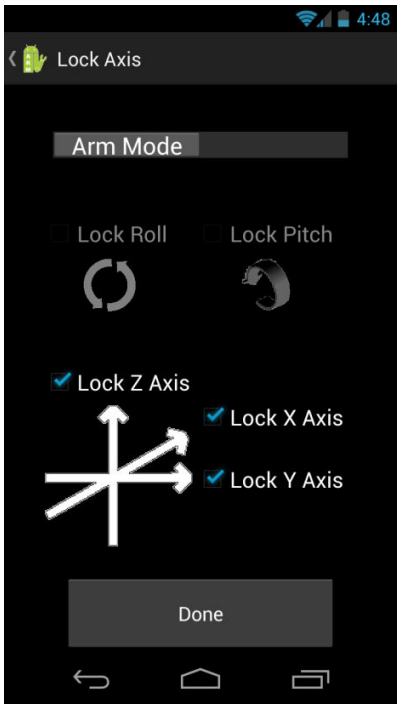

Figure 11: Lock activity (Arm Mode)

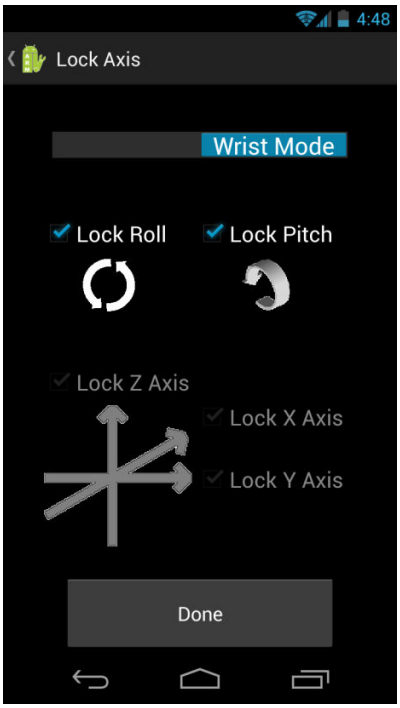

Figure 12: Lock activity (Wrist Mode)

Key Learning

During the development of this project, there are a couple of things that could have been done differently.

The first key learning is the server architecture. The current implementation uses embedded Java code inside MATLAB to act as a server to communicate with the Android app on the smartphone. However MATLAB tends to have non-trivial delays and can sometimes miss a message from the smartphone. This problem is probably due to the poor integration between the server code in Java and the robotic arm controller program in MATLAB. It would likely be better if the server was written as a piece of standalone software in Java or C. Then the communication between the server and the A.R.M. application on the smartphone would be more secure and reliable.

The second key learning is the communication method. The current implementation uses Wi-Fi. Although it is a good standard for transmitting large files, it is not widely supported by embedded devices and thus requires a computer to be connected to the robotic arm. There was an identified problem in loss of data packets and other communication methods may not suffer from this disadvantage. One possibility would have been to use the Bluetooth protocol. This way the communication is strictly one to one and would not be disrupted by other activities. Having both methods of communication implemented can have well-rounded support for many different setups and situations, allowing for more versatile implementations of the application.

Contributions of Each Member

Paul Grouchy

Paul's major contributions to the A.R.M project were all GUI design and development, TCP/IP (i.e. Wi-Fi) communication on the mobile device side and all aspects of the playback/record functionality. Furthermore, he contributed to user experience (UX) through implementing features such as broken Wi-Fi connection detection, vibration features (including haptic feedback on the mobile device side) and auditory feedback and prompts. Finally, Paul implemented axis movement locking and the Wrist Mode/Arm Mode functionality on the device.

René Rail-Ip

As the Apper of the A.R.M. group, René took responsibility on the server side of the system. A server connection had to be built with which the smartphone could connect and communicate. In addition, software that computes the necessary information to control the robotic manipulator had to be developed. In receiving relative displacement coordinates from the smartphone application, a process called inverse kinematics had to be used to transform the coordinates into corresponding joint angles that bring the arm's end effector to the provided coordinates. Furthermore, the server had to distinguish between different modes, home and disconnect commands, and the opening and closing of the gripper, while also testing conditions to determine when to send "Movement Complete" and "Motor Boundary" signals.

Hao Yan

As one of the programmers in the team, Hao coded the modules of the A.R.M. app related to sensors and integration algorithms. Hao also developed calibration and sensitivity features. In addition, he implemented the rotation matrix to transform the acceleration data from the smartphone's coordinates to the absolute coordinates. Finally, Hao has contributed to testing and ultimately enhancing the responsiveness and accuracy of the robotic arm movements by reducing the noises in sensors and improving the integration algorithms.

Future Work

Although all major goals set for the project have been completed, there are some extra functionalities and improvements that could be implemented if work on the project were to continue.

One major improvement would be to include save and load functionality for the record/playback feature. This way the app could keep records of many different movements and the user could then easily choose a suitable movement from the list to playback anytime.

Additionally, further reduction of noise in sensors would greatly improve the intuitive nature of the app. By using more advanced algorithms and filters, such as Kalman Filters, it would be possible to improve the precision of integration results and make the robotic arm movements more smooth and accurate.

Moreover, to ensure the robotic arm moves to the exact position the user wants, a feedback controller for the robotic arm should be implemented. This feedback controller would receive actual encoder readings from the robotic arm to determine if the arm's motion is finished and if the final position is close enough to the position the user required. In the case that the robotic arm fails to respond to the user instructions or reach the desired position, the feedback controller could continuously monitors the state and correct it and could eventually report possible motor errors, such as voltage, position, or torque limits, to the user after a timeout period.

Finally, the app could be extended to be compatible with other robotic and mobile devices. Improved compatibility would allow for a larger user base and a variety of different applications for A.R.M.

Extra

1. Would you interested in having a business school class on marketing/entrepreneurship take it up?
   No, because this project is mostly proof-of-concept.
2. We would like to have our code be available as open source on the Internet.

Resources

1. Coordinates Transformation of acceleration:
   a. http://stackoverflow.com/questions/14963190/calculate-acceleration-in-reference-to-true-north/14988559#14988559
2. Issues surrounding multiple ASyncTasks:
   a. http://foo.jasonhudgins.com/2010/05/limitations-of-asynctask.html
   b. http://stackoverflow.com/questions/11241600/async-task-doinbackground-not-performed
   c. http://stackoverflow.com/questions/4068984/running-multiple-asynctasks-at-the-same-time-not-possible
3. Dynamixel API for Controlling AX12A Motors:
   a. http://support.robotis.com/en/software/dynamixel_sdk/api_reference.htm
4. Java TCP/IP Server Implementation in MATLAB:
   a. http://iheartmatlab.blogspot.ca/2008/08/tcpip-socket-communications-in-matlab.html