David Histing

Shivraj Dalvi

Daniel Thames

Project Report

A system which consists of an operator, computational cluster and controlled object is called a real-time system. The behaviour of such system not only depends on the logical results of computation but also on the physical instant at which these results are produced. The current system that our team is developing consists of a human as an operator, an Android phone running an app that controls a radio-controlled (RC) car as the interface node and computational cluster node and the car as the controlled object. Editing a script command file (or passing messages to the app) through an internet connection will be the man-machine interface between operator and computational cluster. Also, the RC car controller will be the instrumentation interface between computational cluster and rc car. The program running on the android phone will react to the stimuli from the rc car and will control the rc car according to the commands received in the form of some text through the internet from an operator. The time-response of the RTS should conform to the requirements of the task at hand. For instance, if a command for ‘move’ is sent, the system should respond while the utility of the ‘move’ command is still valid. A video feed provides monitoring capability via an internet connected device such as a laptop. In future iterations the rover will utilize GPS commands, whose coordinates will need to be valid within the temporal response of the system. A log file will keep track of geographical location so that the user can track the rover’s performance and diagnose problems.

The Android phone, as the interface node to the computational cluster, has access to a number of advanced hardware features and applications that can assist in controlling the real-time system. The Android phone is interfaced with an Arduino

microcontroller to control the servo hardware directly along with sensors, hardware user controls (knobs, etc.), and operative state indicators (LEDs). The core software of the RTS is a modified, open-source Android application, AudioSerial Out which sends serial information through the headphone jack audio system, which is then converted via legacy modem technology to communicate with the Arduino. Because Android was designed to be a cloud-based operating environment, there are a number of ways to access the device remotely, one of which utilizes a traditional Unix-like shell. From the remote shell, commands can be sent either directly to the application or to a command script file that the application reads and sends to the microcontroller. The video-conferencing application, Skype, is used to monitor the vehicle during remote operation. Finally, GPS tracking information is logged by the Android computational subsystem in .gpx files stored locally. These files can be accessed by a remote user to manually track the location of the rover. There also exists applications that automatically track the location of an Android phone, however, some further app development will be needed to provide geographic navigation that is both more convenient and less cumbersome.

Development

Our team's initial attempt at creating the rover consisted of a toy class RC car base which was modified to interface with an Arduino microcontroller. We immediately ran into issues however, the biggest being the design of the drive system for this particular chassis. The drive for this model was only made to accept commands such as: full speed forward, reverse, and turn while moving commands, which we found to be too crude and the drive system too underpowered to perform in a way that allowed our rover to maneuver accurately. Mainly the rover could not effectively turn in place or maintain a reliable speed and or turning radius. The initial development of this rover was divided thusly: coding and interfacing of the rover was handled by David and Raj while modification to the vehicle was done by Daniel. Upon initial

testing we immediately saw that the platform was simply inadequate and would not produce desired results, leading to our second design.

The second (and most successful) rover design utilized a hobby-class radio controlled car which was interfaced with the same Arduino controller and Android phone system. The car used as the platform was a Team Associated TC3 (Touring Car Version 3) which is a low-slung onroad vehicle made to handle smooth road-like surfaces only. This constraint limited our rover's terrain compatibility, but was the most straightforward design choice given time constraints.

Utilizing this new chassis we were quickly able to get both steering and throttle control working as needed. There was however a slight struggle to get the ESC (electronic speed control) to properly respond to the input from the Arduino. The issue being that the speed control has an inbuilt safety which will not allow it to arm unless a neutral throttle position is detected beforehand. Finding this neutral position turned out to be a tedious guess and check process but was finally cracked by David. Once these problems were overcome we began to add additional sensory inputs and mount the hardware securely to the vehicle. Added soon after initial testing was a sonar input to allow the vehicle to sense obstacles in its path and an indicator light visible on the onboard camera to show when the vehicle was stuck or obstructed. This design load was distributed as follows: Raj and David handled the conversion of the input and drive system over to the new vehicle while Daniel provided the platform and helped to secure and test the new sensors and sensor platforms.

Faults

Throughout the course of development the team encountered numerous faults that had to be addressed. The Android-based subsystem or interface node was the core component of our system and represents the main effort of our project. Due to the emphasis we placed on developing this system and its importance in the total system, it was crucial that it operated with

a high degree of availability and reliability. To this end, it also needed to have a solid method of fault recovery, so that the sub-system could be reset if needed. One of the main faults to overcome manifested in the form of a software bug. Initially, we attempted to use a script file-reading implementation, where a remote shell sent commands to a script file which was then read by the Android device, processed, and then sent to the Arduino microcontroller as serial input. But each file read/write generated a file header information object within the Android operating system, leading to a memory leak that caused premature termination of the AudioSerialOut program. This approach eventually had to replaced with sending remote commands that instantiated the AudioSerialOut program and sent the serial messages altogether. While there are still some issues with this solution, some of which we cannot completely determine (likely due to process pre-emption and priority issues), it is by far more stable and reliable. Due to the robustness of the Unix shell, terminating and restarting the AudioSerialOut process was an adequate fix for the problem. Hardware faults were also a major hurdle to overcome, particularly with the first revision of our chassis, the Tyco RC car. This vehicle featured a passive third wheel which, using the included transmitter, was able to turn fairly well on most surfaces. But, when using the Arduino to send digital signals to the receiver the motors did not produce the same amount of torque. Combined with the friction of the passive wheel on rough surfaces, this prevented the vehicle from turning effectively. To sidestep the issue with attempted replacing the wheel, re-surfacing it, and even elevating the entire back-end of the chassis, but none of these worked very well. This is what led us to use the hobby-class chassis. The new chassis was a much better solution overall, but had too much power and speed for the types of delay that we encountered with network interfaces. Since both the remote network control and video monitoring both had roughly 100 ms of delay, high speed operation was difficult without crashing into objects.

Future Development

In the future we would like to continue developing the concept of a geo-locating rover. Through the knowledge we gained from testing, particularly with physical environments and network performance, it should be possible to produce a rover capable of geo-location on various surfaces with or without obstacles. Since the core computational cluster and interface node are operating with a high availability and reliability, the main problems to solve are related to physical systems or chassis features (correct wheels, suspension, etc.). The great thing about the work we accomplished is that it is highly composable and scalable. Microcontrollers can control just about anything, so through our remote interface we can have this level of control virtually anywhere in the world with an internet connection and adequate network latency/jitter. Overall, we are very proud of the work we did and look forward to developing both the rover and other projects armed with our Android-based remote interface node and knowledge of RTS systems gained from developing the rover.