A Control System for Teams of Off-the-Shelf Scouts and MegaScouts

Bennett Jackson Ian Burt Bradley Kratochvil Nikolaos Papanikolopoulos*
Center for Distributed Robotics, University of Minnesota

***Abstract*-- Two new robotic platforms, the MegaScout and the Commercial Off-the-Shelf Scout (COTS Scout), have been developed recently at the University of Minnesota's Center for Distributed Robotics. The two systems were initially unable to communicate with each other. This paper discusses the development of a communications system and a novel software architecture designed to allow the two systems to work in autonomous cooperation.**

***Index terms*—robotic teams, mobile robotics, miniature robots**

I. Introduction

The new additions to the family of Scout [2,3,7] robots were designed with different goals. The low cost, low weight and small size of new COTS Scout (Figure 1) make it an appealing platform for many kinds of exploration and surveillance. However, its lack of onboard processing power and its analog-only communications have limited its domain of applications to those where it is operated by a human. The new MegaScout platform has a convenient programming environment for onboard processing, high ground speed and reliable digital communications, but its relatively high cost, weight, and size make it less flexible than the COTS Scout for many purposes. By providing a mechanism for digital, computer-based control of the COTS Scout, cooperative groups can be composed from several COTS Scouts and one MegaScout. Once this is possible, many of the advantages of both systems can be combined into a single exploration team. Previous work with heterogeneous teams of robots [1,4,6] has shown encouraging results.

The effort to develop such cooperative units proceeded in two distinct phases. First, at the lowest level, it was necessary to establish a computer-controllable communications link between the two types of robots, allowing the MegaScout to transmit commands to the COTS Scout. This communications link was built in such a way as to make future development with similar components as easy as possible. Next, access to this communications link was made more convenient and more standard by means of a wrapper API designed to work with libUMNRobot, a novel software model for abstracting access to robotic components that was initially developed to support the MegaScouts. This paper will first address each development phase and then present the results of testing and some possible future work, and conclusions.

Figure 1: A COTS Scout with a commercially available RC controller.

II. Phase One: Establishing a Communications Link

The COTS Scouts are equipped with two onboard communications systems. First, a video transmission system broadcasts an analog video stream from the robot's single camera over a fixed FM band. A receiver attached to a teleoperation unit, a mobile or desktop control computer, or another more sophisticated robot can make the video available for viewing or vision processing. In the case where a MegaScout receives the video transmission, existing code makes the individual image frames available to software developers writing vision algorithms.

The second communications system on the COTS Scout is a RC receiver of the type found in many hobby cars and airplanes. This system receives commands from a commercially available, hand-held joystick containing a corresponding transmitter like those shown in Figure 2. This joystick can be used by a human, in cooperation with a

*Author to whom correspondence should be addressed.

video display, to operate the robot remotely. The transmitter relays the two-dimensional position of two different joysticks to the robot by merging four data channels into a single signal. As summarized in Table 1,

Channel	Purpose
1	Speed of servo 1
2	Speed of servo 2 (reversed)
3	Video transmission on/off
4	Not used.

Table 1: Use of each RC data channel by the COTS Scout. only three of the channels are currently in use on the COTS Scout.

A. RC Signal Specifications

The signal is a continuously repeating binary waveform, one cycle of which is shown in Figure 3. The input signal consists of some number of low periods whose widths indicate the values of each of the data channels and which are separated by brief high periods of a small, constant length, as described in [3]. The data portion of the signal is followed by a relatively long low period used to help the receiver identify the beginning and end of each data frame and to stay synchronized with the transmitter. The length of each high period is 500µs. The length of each low periods plus the preceding high period varies between 800µs and 2200µs. The framing gap is 15ms.

Although Figure 1 shows three data channels, any number from one to six may be used. The receiver increments a counter each time a data channel is received and resets that counter when a framing gap is received. In this way, it can track an arbitrary number of channels.

B. A Signal Generation Application

Development of a digitally controlled device for commanding the COTS Scouts was based on several requirements. The system should be reliable and provide the fastest possible response time. It should be as easy to use as possible both for COTS Scout communications as well as for any future development based on similar components. Support for multiple transmitters in simultaneous communication with multiple robots would be preferred. Above all, it must be able to produce a high-resolution version of the signal described above.

A PIC16F877 microcontroller, shown in Figure 4, with a 20MHz oscillator was determined to be sufficient for the demands of the requirements [5]. The first step was a proof of concept task: to generate an input signal to a transmitter using hard-coded values and verify that a robot responded with correct corresponding action. The task was accomplished by programming the PIC in assembly language. Six one byte file registers store the desired output values. A continuous loop uses the PIC's timers to set a digital output pin high or low according to the signal specification. A resolution of approximately one microsecond was achieved, which is more than sufficient. A visually correct signal was observed on an oscilloscope. Attaching a transmitter input to the PIC's digital output showed that a robot responded correctly. By assembling the program with different channel values, the robot could be made to move at different speeds.

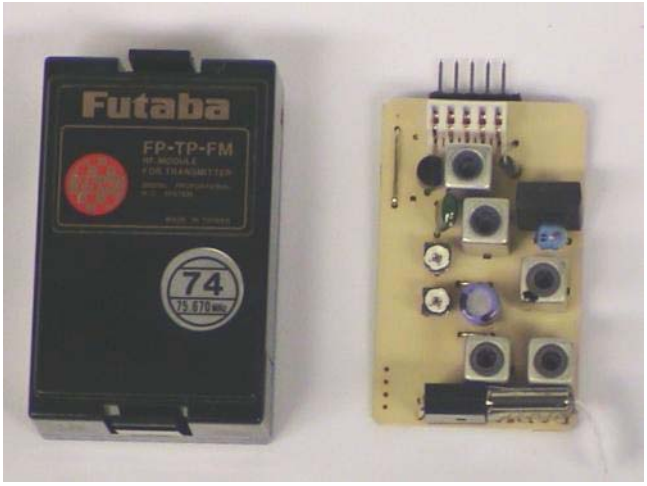

Figure 2: Two RC transmitters. On the left there is a transmitter in its original casing while on the right there is the circuit board inside.

Once the potential effectiveness of the solution had been demonstrated, it was necessary to establish a secondary communication link between the PIC and a PC so that updated values could be sent to change the behavior of the robot in real time. In a final application, the PC might be either a desktop machine or the primary processor of a MegaScout. Because it was desirable to select a common

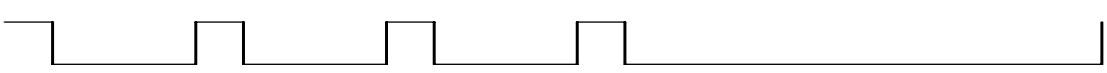

Figure 3a: Transmitter input signal.

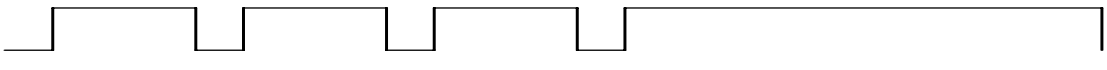

Figure 3b: Transmitter output.

communications standard and because of an existing IIC network in the MegaScouts (based on the BrainStem board, made by Acroname), it was decided that a simple implementation of the IIC protocol would be appropriate.

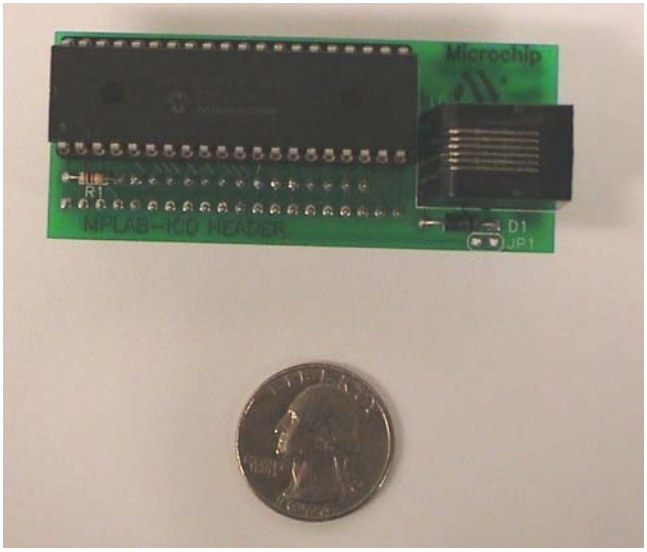

Figure 4: A PIC16F877 prototyping chip with in-circuit debugging hardware. Versions used in the final products are much smaller.

IIC (often written i2c) stands for Inter-Integrated Circuit. It is a very simple, two-line, serial communications protocol. Networks consist of a single bus supporting up to 126 devices (more in some versions) at clock speeds of 100KHz, 400KHz, and 1MHz. Each network has a single master device and any number of slave devices. The master is responsible for establishing the regular pulse on the clock line and mitigating collisions on the data line. Many devices, including many commonly available robotic sensors, come equipped for IIC communication.

The protocol provides a simple, byte-oriented packet structure. First, a start signal is sent by a transmitting device, indicating that bytes are about to be sent. The first byte of every packet is the seven-bit address of the target device and a bit indicating whether this is a request to initiate a read operation from the target or whether more bytes will follow that should be written to the target. The target machine must send an ACK pulse after each byte. In the case of a write operation (the only portion of the standard implemented for this project), one or more data bytes follow the address. Finally, the transmitter sends an end pulse to mark the conclusion of the packet.

The PIC provides a convenient, interrupt-driven implementation of IIC. After configuring all appropriate settings, a PIC assembly program that is part of a slave receiver need only have a single interrupt handling routine that will be called each time a byte has been fully received.

The packet structure chosen for this project is illustrated in Figure 6. All correctly formed packets consist of four bytes. First, according to the IIC protocol is the device's address. Next is the ID of a particular transmitter. Though not yet implemented, this will allow two or more

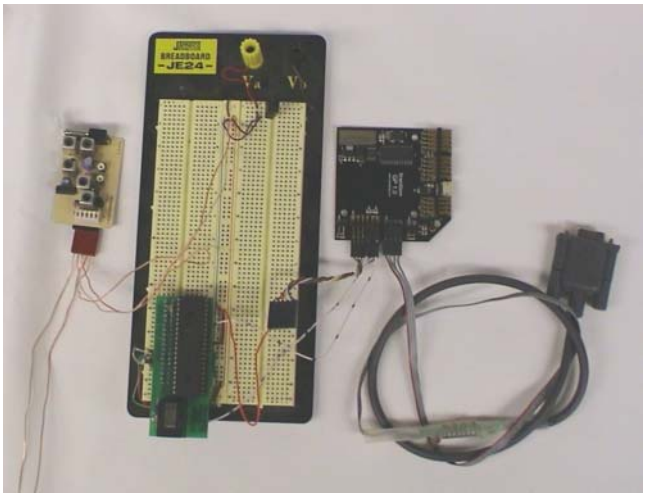

Figure 5: Final prototyping circuit showing the transmitter, PIC, and BrainStem fully connected for IIC communications.

transmitters to be controlled simultaneously by the same PIC, making larger robotic teams more practical. The third byte is the number (between 0 and 5) of the data channel being changed. Finally, the last byte gives the new value to be transmitted on that channel. A value of 0 will cause the shortest possible signal (about 0.8ms) and a value of 255 will cause the longest possible signal (about 2.2ms). After completion, the IIC communications system was tested by connecting it to the IIC pins of a BrainStem, and manually sending packets from the BrainStem's console application, successfully, though awkwardly, controlling a robot.

The result of the PIC development is a fully functional communication link from a PC to a COTS Scout. The system is fast, built on industry standards, cheap to assemble, and easy to use. It can be easily expanded to support simultaneous control of multiple robots.

III. Phase Two: Wrapper API

A. *The libUMNRobot Framework*

The libUMNRobot framework is a novel suite of object-oriented programming tools written in C++ and designed to provide a convenient, uniform interface to a wide variety of robotic components. The library was originally written to support the MegaScout platform, but with the additional goal that it should support multiple, heterogeneous robotic platforms with little or no modification. Use of the framework to support the COTS Scout is the first test of this heterogeneous use. The framework provides a mechanism for enumerating all the components of a robot and all the configuration and calibration properties of each component in a single, hierarchical XML document. This

makes it very easy to express the subtle differences between large numbers of similar robots without having to make any modifications to the control code for those robots.

A greatly simplified UML diagram of the object model for the framework can be seen in Figure 7. The framework is a set of loosely coupled libraries, each with a specific goal.

[ADDR]	[XMIT ID]	[CHAN ID]	[DATA]

Figure 6: Packet structure expected by PIC consists of four bytes: The IIC address of the PIC, the ID of a transmitter, the ID of the data channel being modified, and the new data value for that channel.

At the core is the UMNRobot library itself. This library defines a generic interface called *Capability* that is the root interface of all robotic components. *Capabilities* can have a set of child *Capabilities*. *Robots* are composed of such hierarchies of *Capability* objects. *Sensor* and *Actuator* are sub-interfaces of *Capability* written as templates. The UMNRobot core library also defines a standard for instantiating arbitrary robots by means of a *RobotFactory*.

An affiliated library, "UMNRobot Xerces Components" provides a concrete implementation of RobotFactory that instantiates Robots from XML descriptor files.

Another library, "UMNRobot Common Components", provides a richer set of generic interfaces that model a wide variety of frequently used robotic sensory and actuation components. This library contains interfaces such as *Motor*, *Servo*, *RangeFinder*, *DigitalSwitch*, *AxelEncoder*, *PhotoelectricSensor*, and *VideoSource*. These interfaces impose a data type and a set of units on the inputs and outputs to their corresponding devices, but they contain no specific code for communicating with actual devices.

Such communication is accomplished by a lower level suite of libraries that provide models for specific hardware. For example, the library "UMNRobot BrainStem Components" contains implementations for controlling many sensors and actuators through the BrainStem board. Similarly, "UMNRobot V4L Components" contains implementations for video inputs that use the Linux video standards.

These libraries are not all intended for use in every robotic application. Instead, each application developer picks the set of libraries appropriate to the hardware in use and only

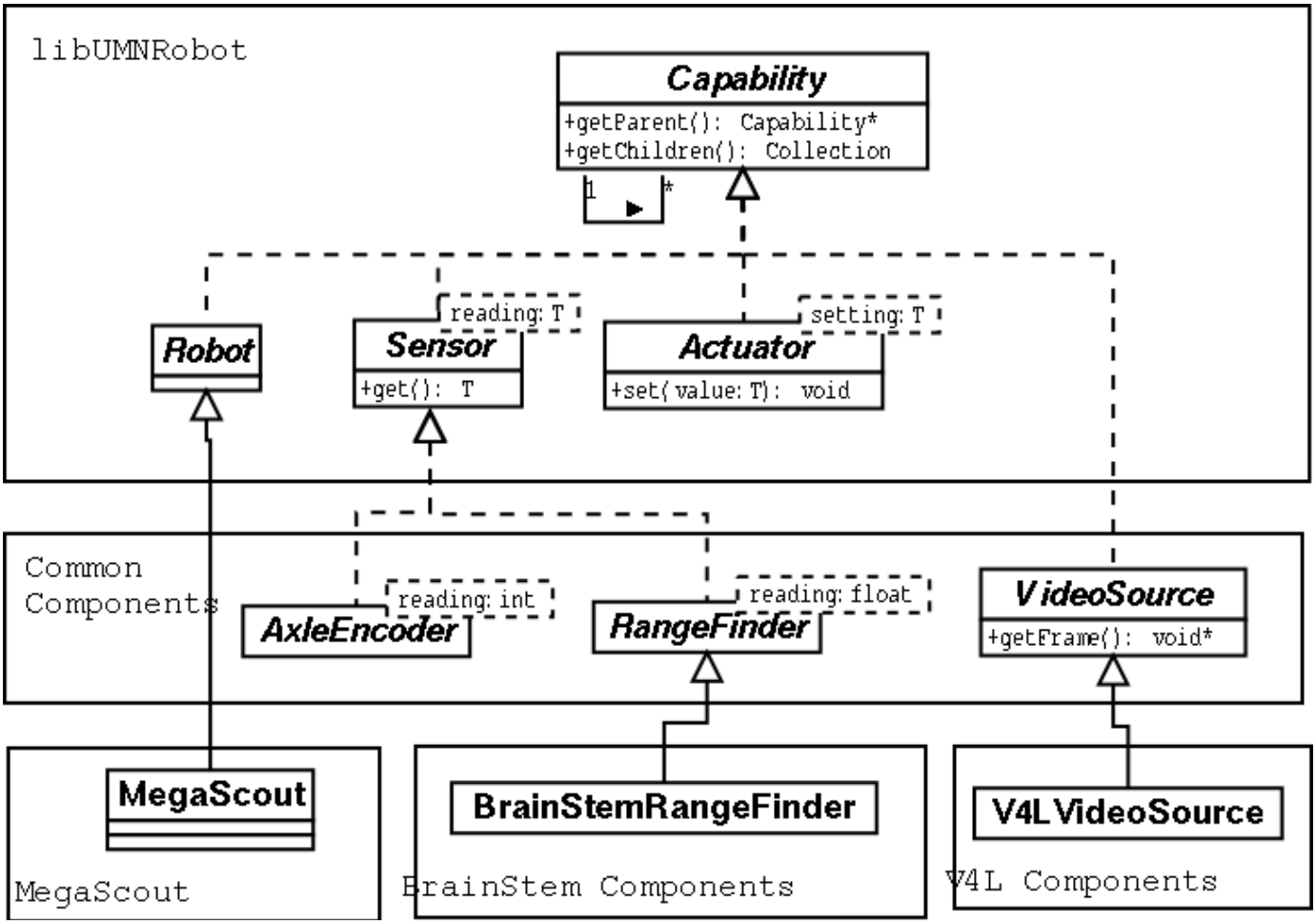

Figure 7: Simplified UML diagram of the libUMNRobot object model.

compile against the relevant implementations. In this way, many robots can use the same generic interfaces for different underlying implementations while maintaining installation footprints that are as small as possible.

B Incorporating the COTS Scout into the UMNRobot framework

Because this framework is the basis for the ongoing development of the MegaScout platform and because there was interest in validating the idea that the same framework could be applied to distinct robotic platforms, it was decided that the new computer control system for the COTS Scout would be written against libUMNRobot. These modifications first required expansion the core libraries and then the addition of a new package dedicated to the COTS Scout. Each step is discussed below.

The UMNRobot core packages were expanded first to contain a generic interface for RC communication, and then a specific implementation based on proxying the IIC packets discussed earlier through a BrainStem. Another implementation could easily be added that communicated with an IIC bus through some other conduit. *BrainStemComponents::RCMotor* derives from *Motor* and *RCDevice*. Similarly, *RCDigitalSwitch* extends *DigitalSwitch* and *RCDevice*. These components can be instantiated from XML descriptors jut like the previously existing components.

The new COTS Scout package relies on the additions to the UMNRobot core and provides an API for instantiating a *COTS Scout::Robot* object through a derivative of RobotFactory. An example of an XML descriptor file for a COTS Scout can be seen in Figure 8, which illustrates all the configurable parameters of each component.

IV. TESTING

As a test for the whole structure that was built in phases one and two, a simple interface was built to allow a human to drive the robot. The application was a text-based application for a Linux console, and it was run on the machine used for prototyping MegaScout software. The program accepts directional input from the user by means of keyboard arrow keys. The space bar provided an immediate stop function.

This application made it possible to drive the robot effectively. It was necessary to calibrate the zero point of the servos for the robot so that a command packet of zero actually caused the corresponding wheel to stop. This calibration was done empirically and the results were

```
<capability type="COTSScout::Robot" name="Robot">

<capability type="UMNRobot::CommonComponents::Wheel" name="PORT_WHEEL">
<parameter name="diameter" type="float" value="8.25" />
  <capability type="UMNRobot::BrainStemComponents::RCMotor" name="MOTOR">
  <parameter name="range-min" type="int" value="-100" />
  <parameter name="range-max" type="int" value="100" />
  <parameter name="pic-i2c-address" type="unsigned int" value="100" />
  <parameter name="transmitter-id" type="unsigned int" value="0" />
  <parameter name="channel-id" type="unsigned int" value="1" />
  <parameter name="invert" type="bool" value="t" />
  <parameter name="offset" type="int" value="140" />
  <parameter name="brainstem-id" type="unsigned int" value="2" />
  </capability>
</capability>

<capability type="UMNRobot::CommonComponents::Wheel" name="STARBOARD_WHEEL">
<parameter name="diameter" type="float" value="8.25" />
  <capability type="UMNRobot::BrainStemComponents::RCMotor" name="MOTOR">
  <parameter name="range-min" type="int" value="-100" />
  <parameter name="range-max" type="int" value="100" />
  <parameter name="pic-i2c-address" type="unsigned int" value="100" />
  <parameter name="transmitter-id" type="unsigned int" value="0" />
  <parameter name="channel-id" type="unsigned int" value="0" />
  <parameter name="invert" type="bool" value="f" />
  <parameter name="offset" type="int" value="120" />
  <parameter name="brainstem-id" type="unsigned int" value="2" />
  </capability>
</capability>

<capability type="UMNRobot::CommonComponents::VideoSource" name="CAMERA">
 <description>Camera</description>
 <parameter name="port" type="unsigned int" value="0" />
 <parameter name="video-set-name" type="string"
  value="FRAMEGRABBER" />
   <capability type="UMNRobot::BrainStemComponents::RCDigtialSwitch" name="CAMERA_POWER_SWITCH">
    <description>Switch that turns the video transmission on and off.</description>
    <parameter name="pic-i2c-address" type="unsigned int" value="100" />
    <parameter name="transmitter-id" type="unsigned int" value="0" />
    <parameter name="channel-id" type="unsigned int" value="2" />
    <parameter name="invert" type="bool" value="f" />
    <parameter name="brainstem-id" type="unsigned int" value="2" />
   </capability>
</capability>

<capability type="UMNRobot::V4LComponents::VideoSet" name="FRAMEGRABBER">
<description>Picasso PC104-2SQ Framegrabber</description>
<parameter name="video-device-number" type="unsigned int" value="0" />
</capability>

</capability>
```

Figure 8: A sample XML descriptor for a COTS Scout object.

specified to the control architecture in the XML descriptor. Due to the nature of the program and the poor precision of the robot's components, the robot turned sluggishly at high speeds and was difficult to drive exactly straight. With more work spent on the algorithm for selecting speeds based on input keystrokes a more natural control interface could be developed.

This application highlighted one of the known problems with the COTS Scout platform: adjustment of wheel trim. The servos used in the robot do not provide a constant output for the same inputs. The data value used to stop the motors shifts over time, and the exact speeds during motion appear to vary slightly as well. The result is that keeping the robot stationary requires constant intervention. When combined with the irregularity and flexibility of the wheels, this arbitrary variation in motor output makes controlled, straight motion difficult to achieve.

V. Future Work

In order for a working MegaScout to control a team of COTS Scouts, it will first be necessary to produce a compact version of the electronics that can be placed in a MegaScout sensor bay. The components are sufficiently small to make this possible.

The availability of a programming API for the COTS Scout brings a range of new applications into reach. Ideas for such applications and specific plans are discussed below.

A WindowsCE teleoperation unit for the MegaScout was recently completed. This system runs on a Compaq iPaq and provides streaming digital video, streaming sensor data and the ability to drive the robot. The interface can now be easily adapted to drive the COTS Scout. Alternatively, the interface could be expanded to allow the user to switch at runtime between the control of a MegaScout and one more COTS Scouts controlled via the MegaScout. This would allow video-based teleoperation of an entire robot team from a single hand-held device.

Many possible autonomous applications for a single COTS Scout exist. A hide-in-darkness behavior similar to the one implemented on the original scouts is possible. A visual feedback loop that automatically adjusts the wheel trim of the robot would be very helpful. Possible applications for teams of robots include recognition of other robots, formation driving, and follow-the-leader. A larger robot like a MegaScout may eventually be equipped with a marsupial system for carrying a fleet of COTS Scouts into an environment. COTS Scouts would then need to be able to explore and return to the base of operations.

VI. Acknowledgements

This material is based upon work supported by Microsoft Corporation, the National Science Foundation through grant #EIA-0224363, and the Defense Advanced Research Projects Agency, Microsystems Technology Office (Distributed Robotics), ARPA Order No. G155, Program Code No. 8H20, issued by DARPA/CMD under Contract #MDA972-98-C-0008.

Everyone at the Center for Distributed Robotics especially Andrew Drenner, Derek Goerke, Colin McMillen, Sascha Stoeter, Kristen Stubbs, and David Waletzko helped in the development COTS Scout and MegaScout platforms. This work would not have been possible without their efforts.

VII. References

[1] Y. U. Cao, A. S. Fukunaga, and A. B. Khang. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots,* 4(1), pp.7 -27, Mar. 1997.

[2] A. Drenner, I. Burt, B. Kratochvil, B. J. Nelson, N. Papanikolopoulos, and K. B. Yesin. Communication and mobility enhancements to the scout robot. *In Proc. of the IEEE//RSJ Int'l Conf. on Intelligent Robots and Systems*, Lausanne, Switzerland, Oct. 2002.

[3] D. F. Hougen, J. C. Bonney, J. R. Budenske, M. Dvorak, M. Gini, D. G. Krantz, F. Malver, B. Nelson, N. Papanikolopoulos, P. E. Rybski, S. A. Stoeter, R. Voyles, and K. B. Yesin. Reconfigurable robots for distributed robotics. In *Government Microcircuit Applications Conf*., pp. 72-75, Anaheim, CA, Mar. 2000.

[4] Y. Ishida, H. Asama, S. Tomita, K. Ozaki, A. Matsumoto, and I. Endo. Functional complement by cooperation of multiple autonomous robots. In *Proc. of the IEEE Int'l Conference on Robotics and Automation*, pp. 2476-2481, 1994.

[5] D. Rus and K. Kotay. Versatility for unknown worlds: Mobile sensors and self-reconfiguring robots. In A. Zelinsky, editor, *Field and Service Robotics.* 1998.

[6] J. Spofford, D. Anhalt, J. Herron, and B. Lapin. Collaborative robotic team design and integration. In *Proceedings of SPIE Unmanned Ground Vehicle Technology II,* Apr. 2000.

[7] S. A. Stoeter, P. E. Rybski, K. N. Stubbs, C. P. McMillen, M. Gini, D. F. Hougen, and N. Papanikolopoulos. "A robot team for surveillance tasks: design and architecture." *Robotics and Autonomous Systems*, Elsevier, vol. 40/2-3, pp. 173-183, September 2002.