A LEGO-based mobile robotic platform for evaluation of parallel control and estimation algorithms

Fredrik Wahlberg, Alexander Medvedev, and Olov Rosén

***Abstract*— An inexpensive robotic system intended for educational use in parallel algorithms for embedded control and signal processing is described. The hardware platform is comprised of a state-of-the-art multi-core system in a wireless network with several mobile LEGO robots that collect data from their environment. The setup covers a broad range of real-time cooperative and parallel problems arising in sensor networks, robotics, surveillance and high-performance embedded applications. As an illustration, a bearings-only tracking problem, estimating both mobile robots positions and the position of a non-cooperating target by using parallel particle filtering, is solved on the proposed platform. In order to improve the estimation accuracy and to adjust to changes in the environment and movements of the target, a controller positioning the mobile robots is utilized.**

***Index Terms*— Control education, Real time, LEGO, Mobile Robots, Tracking, Multi-core, Sensor Networks**

I. Introduction

Processor manufacturers are to an ever increasing degree using multi-core approaches, instead of only increasing the clock frequency, to achieve higher computing power. An advantage with the multi-core approach in embedded systems is the increased computing power per unit energy, conserving battery. This computing power does not necessarily mean that demanding real-time applications of today will be running faster or spend less energy on a multi-core computer.

Most of the algorithms in automatic control and signal processing are developed to run sequentially with full access to sensor data and are not capable of taking advantage of the parallel environment. Combined with real-time process scheduling, the effect of running on a multi-core system can even be a reduction in execution speed compared to a single core processor system with a comparable clock frequency. If an algorithm assumes access to information that is not instantly available in cache memory, valuable computing time amounting to hundreds of processor cycles might be lost while waiting for it.

Similarly, in such distributed applications as sensor networks, the estimation problem might be spread over several physically separated nodes with only partial network connectivity. Here the algorithms need not only be parallel but also to take the network topology and communication costs into account.

Developing new and modifying existing real-time algorithms so that they are aware of the computational platform and take advantage of parallel and distributed computing is therefore an important and sometimes challenging task.

A clear tendency in control curriculum of the last decade is the introduction of courses dealing with real-time systems and embedded control. These make typically use of a concoction of computer science and control engineering approaches. In view of the transition to multi-core embedded platforms, control courses have to face the challenges and embrace the possibilities of parallel implementation of control and estimation algorithms. At the same time, in an academic environment, it not always easy to come up with a well-defined and sufficiently rich example of a control application that would illustrate the benefits and pitfalls of parallel implementation.

The robotic system described below is intended for development, testing and benchmarking of parallel algorithms in advanced automatic control and signal processing courses, both in laboratory and project assignment formats. As an illustration, a bearings-only tracking problem solved by parallel particle filtering is worked out in detail.

In a so called bearings-only problem, the only available measurements are the angles to a moving target, relative some reference angle from a sensor with a known position. The bearings-only problems are inherently non-linear with observability issues and are typically solved with respect to a set of constraints on the state vector range.

The mobile sensor network is built using three LEGO Mindstorms robots. Sensor networks consisting of mobile robots are a well explored area with many commercial applications available [1], [2], [3], [4], [5]. The Mindstorms kit is fairly cheap and contains a fully functional robotic system that is easy to use and re-use. Video cameras mounted on mobile sensor nodes were the only sensors used, giving a realistic data for the estimation problem.

The server is a state-of-the-art multi-core machine using 12 processor cores to perform state estimation of the sensor network and the non-cooperating tracked object. The price of the advanced server is still just a fraction of that of an off-the-shelf laboratory unit. Data processing is handled both in a distributed manner on the sensor nodes and on the central server, also acting as network hub.

In Section II, a brief overview of the hardware platform is provided while system implementation is described in

The authors are partially supported by the project "Computationally Demanding Real-Time Applications on Multi-core Platforms" funded by Swedish Foundation for Strategic Research.

F. Wahlberg, A. Medvedev, and O. Rosén are with the Department of Information Technology, Uppsala University, Uppsala SE-751 05, Sweden, email: jf.wahlberg@gmail.com, {olov.rosen, alexander.medvedev}@it.uu.se

Section III. In Section IV, some experimental results are presented to exemplify the implemented platform's capabilities. Finally, in Section V, the advantages and disadvantages of the implemented platform and LEGO Mindstorms are summarized.

II. Hardware overview

A. Sensor nodes

The mobile sensor nodes are built using the LEGO Mindstorms robotics kit, [6]. It consists of a computer block (referred to as the NXT) and several basic sensors and actuators described below. The NXT is equipped with a 32-bit ARM processor with a clock frequency of 48 MHz, 256 kB flash memory, and 64 kB RAM memory. It has 3 output ports for actuators and 4 input ports for sensors, [7].

1) Operating system: The operating system shipped by LEGO is capable of receiving and executing simple commands via bluetooth. Minor programs can be downloaded to the NXT using a graphical user interface for constructing simple algorithms comprised of block schemes with predefined blocks. However, if more advanced algorithms are to be implemented, there are many alternative operating systems that can be used. Most of them have a basic micro kernel with real time capabilities that can run user-defined software (usually written in C/C++).

One OS stands out: a Java virtual machine available as an open source project called leJOS, [8]. It enables the user to implement and run on the NXT any algorithm in Java with an almost a full set of Java libraries. The use of the Java language makes the development faster and simpler than in C/C++, with the extra advantage of being suitable for educational purposes. All interfaces for sensors/actuators are implemented in the standard (object oriented) Java way with an interface object accessible to the programmer as a library component. The downside of using Java is the reduced execution time compared to C/C++. leJOS has been selected for the implementation described below.

2) Communication: The NXT is equipped with an USB port and bluetooth. Communication over USB is limited by the length of the cable, so the main way of communication is over a bluetooth link. All network communication is transmitted using the RFCOMM protocol emulating an RS232 connection between the devices. The network two-way latency was observed to be lower than 40 ms, using one device. If more devices are used, the latency increases somewhat but usually stays below 70 ms. Latency increases even more if the network strain is high.

The range of bluetooth can be a problem. Some low-power implementations have seriously degraded performance already at a couple of meters. Using a more powerful antenna (class 1), the effective range can be extended to more than 20 meters (100 meters is the theoretical maximum in the specifications).

3) Sensor and actuators: The Mindstorms kit is delivered with a basic set of sensor and actuators such as three servo motors and sensors for touch, sound, ultrasound and light intensity (a one pixel camera). There is however a range of third party devices available including multiplexers for attaching extra sensor or actuators to one NXT.

Light sensors come in many flavours. The basic one has just one grey scale pixel but there is also an official version of it that detects colors. For more advanced needs, there is a sensor array and a full video camera. The sensor array is simply several parallel light sensors on a decimeter long plate to detect difference in surface brightness along a line. The low resolution camera is attached to a micro-controller to process the image since the sensor bus is too slow to upload the full image to the main processor. Parameterized data are however accessible from the micro-controller.

Range finders exploiting either laser or ultrasound are available. The accuracy, resilience to different target geometries, and range varies. Some sensors of this type have many range finders mounted on one unit.

Motors controlled by pulse-width modulation are fitted with tachometers yielding an accuracy of one degree of the angles turned from the zero point. More and less powerful ones than the official LEGO motors are available.

Touch sensors are only binary but can still be used for collision detection or as a button.

Linear actuators can extend a rod able to push objects.

Gyroscope and Accelerometer are used to estimate a change in position or the direction of gravity. The accuracy is very high and these sensors are capable of keeping track of a moving robot for an extended period.

Sound sensor can record audible sound well enough for transmitting data via sound waves.

Compass and GPS sensors can position the robot as good as any smartphone.

B. Server

The server used is a HP Z800 workstation with two 6 core processors with 384 kB of L1 cache, 1536 kB of L2 cache and 12288 kB of L3 cache per processor. Being able to test algorithms on as many cores as 12 is important since the advantages of parallelism and such effects as false sharing or super linear speedup can be better observed with more parallel processes running. Embedded platforms in the future will probably contain many more cores than 12, but the present solution makes a sufficient test case for algorithm scalability.

III. Implementation

A. Problem description

As an illustration, the mobile sensor nodes, together with the server, are set to solve a bearings-only tracking problem. The main task is to estimate the position of a non-cooperating target out of angle measurements (bearings) obtained by the mobile robots. Measurements are obtained by the video cameras attached to micro-controllers for image analysis (the so-called NXTCams). To estimate the position of the sensor nodes, the robots move inside a predefined space (called the arena) with landmarks positioned at given locations, as shown in Fig. 1. A regulator issuing movement commands to the robots is used to increase the accuracy of

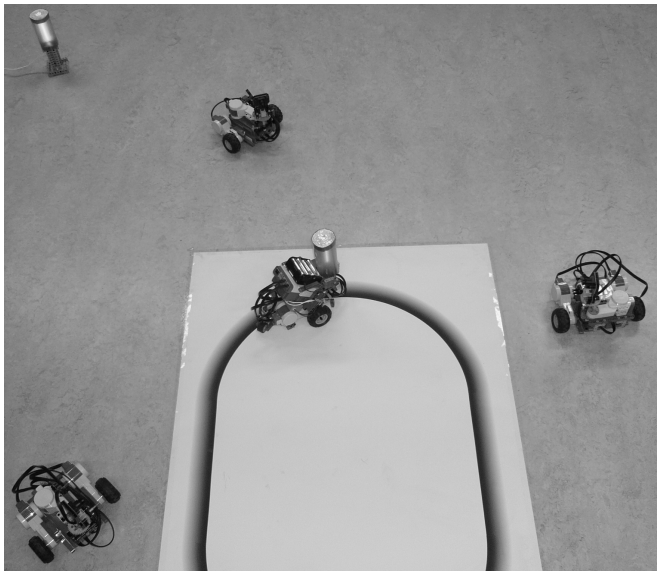

Fig. 1. Overview of a part of the arena showing the non-cooperating target following a path surrounded by three sensor nodes. At the upper left corner, a landmark can be seen.

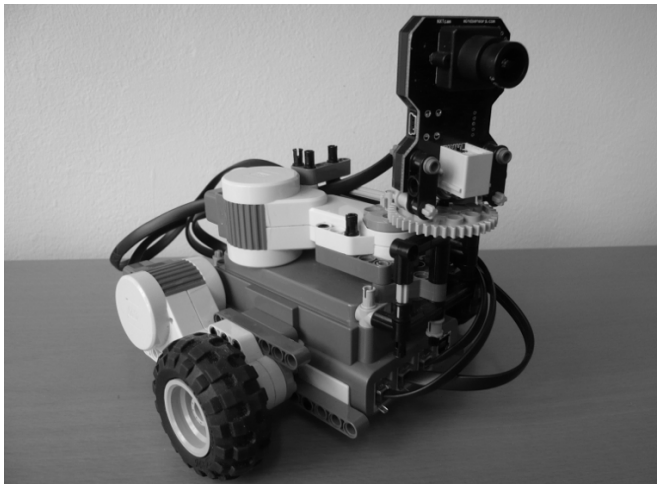

Fig. 2. Mobile robot with actuated video camera (sensor node)

the information collected by each sensor. The implemented system is described in more detail in [9].

B. System design

Of the available hardware, three motors and one camera are used for building each robot, as shown in Fig. 2. The camera is used to locate the target and the landmarks. Two of the motors move the robot and one motor is used to actuate the camera. Each motor has a tachometer providing information on distance travelled, angles turned, and camera heading.

The server software is written in Java and has a simple graphical interface through which movement commands could be issued and internal data could be viewed. Ubuntu Linux is used as operating system.

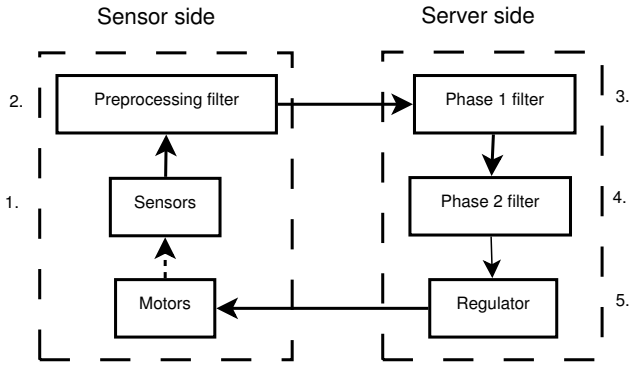

Fig. 3. The implemented system from a control loop perspective (section III-D)

C. Development strategy

A disadvantage of the Mindstorms platform under leJOS is the lack of relevant debugging tools. This would not have been a big problem if some unsophisticated software were to be implemented. However, the developed system needs multiple threads on the NXT and an advanced data processing pipeline on the server. Uploading the compiled code to the sensor node took about 25 seconds and the camera heading would have to be reset manually.

To cut down on the development time, a simulation environment was implemented where filtering and regulator components could be tested and debugged before being uploaded to the robots. This simulation environment only served as a debugging tool and not for data collection. When a component could be considered mature enough, the code was copied into the production repositories.

D. The control loop

A view of the system as a networked control loop is shown in Fig. 3. The link between the motor and sensor block in the figure is dashed indicating that the motors influence the sensors ability to collect data and in that way complete the loop. Solid arrows indicate direct communication. The block numbers in Fig. 3 correspond to the numbers in the following subsections.

1) Data acquisition: The sensors of the mobile robots collect data on their environment and on motor status. This provides information on the bearing to a tracked object (landmark or robot) in the sensors own reference system. The robots also measure the angles turned and the distance travelled. The data from the motor and camera controller thread are packaged together, time stamped, and pushed to an internal buffer. The buffer is sorted according to the time stamps of the data packets.

2) Preprocessing: The preprocessor reads packages from the internal buffer at even intervals and prepares the sensor data for upload to the server. Data packages in the buffer have high temporal resolution that is not requested on the server side. To decrease network strain, information is aggregated into fewer packages. Finally, data from the preprocessing filter are serialized and written to the network stream.

3) Filtering (first phase): The server receives the data and after deserialization passes it to the first phase of filtering. There is one instance of this filter for each sensor node. These filters estimate the position of each node relative to the known landmarks as a part of the state vector consisting of the position in the arena plane and the heading angle. Estimations are performed by a particle filter and also provide the certainty of the estimated state vector evaluated through the estimated probability distributions.

4) Filtering (second phase): The second filtering phase uses information from all the sensor nodes to find the position and velocity of the tracked target. It is implemented using a particle filter in an external process. The filtering in the external process is not constrained by the Java virtual machine and can utilize the full power of the multi-core server.

5) Regulation: If the certainty in the estimation of the first and second filter phases is not high enough, the regulator adjusts the positions of the mobile robots by commanding them to move. The commands contain information on the sensor nodes position and where to go.

A regulator has been implemented that uses a set of rules to try to move each robot to increase the information acquired by the whole system. These rules were set to move the robots to surround the target and be close enough for good camera readings but still not risk a collision (see Section III-G).

Each of the steps 2-5 is implemented as a separate component inheriting a base class for each type of problem. The base class provided interfaces to the system to facilitate further development of new components.

E. Communication

1) Data transmission: Communication between the sensor nodes and the server is implemented via bluetooth. Data are put in container classes and serialized to a byte array that is transmitted. All communication is performed through the server.

2) Clock synchronisation: It was obvious from the beginning of the project that a shared time reference would be needed. This is a common problem in distributed systems. The standard approach used in time synchronization protocols, such as the Berkeley and NTP protocols, is to use Christian's algorithm [10]. In Christian's algorithm, it is assumed that the transmission time of sending a request from one network node to another is approximately the same as the transmission time of receiving the answer, and that the request processing time is much lower than the latency. These are valid assumptions here since there are only a few nodes on the piconet (bluetooth network) used.

F. Filtering

Several particle filtering methods have been implemented, some of them in native Java and some in C++ with OpenMP for parallelism. The parallel particle filters implemented in C++ are described in [11].

1) Preprocessing: A preprocessing filter on the NXT was implemented to reduce network traffic and to remove excessive noise. Sometimes the camera would see something in its environment, for instance a reflection of the sun, and interpret it as a landmark. These erroneous bearings lead to faulty sensor position estimates and have to be removed. To classify which bearings to keep, the robot keeps track of its estimated position. The local estimate is updated every time a regulator command is issued. By knowing the robot's own position and the landmark positions, a bearing resulting in a position estimate that is too far from the previous value could thus be ignored. This has proven itself to be a viable strategy to reduce extreme noise to a reasonable level.

2) Filtering pipeline: The first phase of filtering estimates the states of each sensor node using the camera and motor information. The estimated states are the position in the $x-y$ plane and the heading angle, all relative to the arena. The sensor nodes try to track their own movements but their estimates could not be considered reliable since they assume information from the motors to be exact. Hence all uploaded data are set in each sensor node's own reference system. The code for the first filtering phase is written in Java and only two filtering techniques have been tested.

The noise distribution in the bearings of landmarks and the target is approximated as Gaussian. This approximation has worked well for tracking and is also confirmed later by studying the actual distributions.

The second phase of filtering estimates the position and velocity of the target using an number of particle filtering techniques. A filtering component is written in Java to spawn an external process. The external process is connected to the virtual machine through the standard input and output pipe of the process. Using this technique, efficient inter-process communication could be set up and allow for filters being implemented in any programming language. The external filters are written in C++ and utilize OpenMP that is a package to facilitate the use of many different threads for computations. The communication protocol is a byte protocol using Google protocol buffers [12] for serialization of data.

3) Methods: Three types of filters have been implemented in C++/OpenMP and later modified to fit the communication model used in the present platform. The filtering methods used are briefly described below, for more details see [11].

All parallelization here is implemented using a manager/worker model where each worker thread is assigned tasks by the manager thread. The manager thread performs all sequential steps.

In the first phase of filtering, no parallelization is used. The following parallel filters have been used in the second phase of filtering:

a) Gaussian particle filter: The Gaussian particle filter assumes the *a priori* and *a posteriori* probability density function of the state variable to be Gaussian. This gives a very simple parametrization for the information carried out from one iteration to the next one. It is also easy for all threads to take part in re-sampling and calculation of the mean. An important advantage with this approach is that, if

correctly implemented, all particles might live in the cache of the processor indifferent of the total amount of particles. When implemented on many physically separate nodes, the network bandwidth demand is kept low since the only data needed to be exchanged is the parameters of the *a posteriori* distribution. The disadvantage being that any significantly non-Gaussian distribution is harder to estimate.

b) Globally distributed particle filter: The globally distributed particle filter takes full advantage of a shared memory environment. All particles can be accessed by any thread at any time. To stop threads from using the same particles, each worker thread is assigned an interval of particles to work with. Propagation and evaluation of the particles can be done in parallel while re-sampling is performed by a manager thread. Any distribution can be represented by this filter but the implementation here is restricted to a unimodal probability distribution.

c) Locally distributed particle filter: The locally distributed particle filter is a method for distributed calculation without a shared memory. Every node creates its own particle set on which to perform propagation, evaluation, re-sampling and calculation of an estimate. To keep the threads together as one filter, each node exchanges a portion of its particles on every iteration with other nodes. Local estimates are transmitted together with a weight to form a global estimate. This approach is suitable when many physically separated networked nodes collectively perform the filtering and a non-Gaussian distribution is being estimated. As with the globally distributed particle filter, any distribution can be represented.

G. Regulator

The most basic approach to the regulator problem was to make a simple set of rules for sensor movement to improve estimation performance. Solving a control problem in this way can be quite tricky since stability and convergence of the resulting regulator are hard to prove. The problem that is solved here is however very intuitive and the achieved control performance is supposed to establish some kind of lower bound by keeping the system functional. More advanced controllers based e.g. on the model-predictive control machinery are expected to exhibit better performance.

A set of rules has been constructed where a desirable distance between the target and each sensor is set and the intersection angles of the sensors line of sight to the target are to be maximized (possible only if there are three or more sensor nodes). Also, any issued command should not be able to move the sensor node outside the arena. This is made possible with a safety buffer at the arena edges, to allow for some errors in the estimated robot positions.

The regulator has to be somewhat conservative with respect to relocation of robots since movements introduce errors in their estimated positions. All movement are subjected to a threshold preventing the sensor nodes from moving often for smaller distances. Every movement, however small, would have to be weighed against the possible advantage of gaining a better view for the camera.

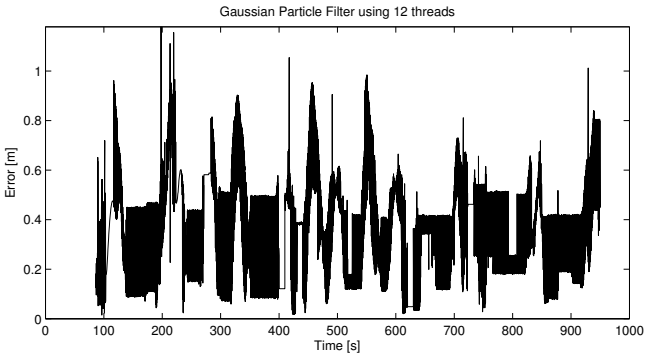

Fig. 4. The mean error, using the Gaussian particle filter, measured as euclidean distance from the reference position to the estimate at a given time. Some regions in the plot seem black since the series is long and the estimate varies significantly. New data was received a few times per second and a new estimate was successfully produced for most of the acquired bearings (Section IV).

IV. Experiments

Experiments were performed to test the implemented platform and demonstrate its viability.

Even though the calibration of the robot motors has been hard to perform (parts of the robots would separate slightly during a data collection session) the estimation of the positions of the robots is satisfactory. The filters are able to compensate for errors both under turning and driving straight. In one test, the robots have been ordered to move around a hexagonal path for an extensive amount of time, which mission they carried out without problem. The errors that were hard to compensate for occurred when the camera gave an erroneous reading at the same time as a sensor node drifted more than usual. In these extreme cases, it could take some time for the sensor nodes to compensate for the errors.

A. Estimation accuracy

The accuracy of the complete setup using the external tracking filters has been tested. The target was a robot tracking a closed contour (running track-like path in the middle of the arena). The regulator was used to prevent the robots from colliding with the target and keep tracking accuracy as high as possible. Three sensor nodes were set to track the target.

The mean accuracy of 50 runs with one data set of the Gaussian particle filter using 12 threads is shown in Fig. 4. The estimation accuracy is affected by both filtering phases and the regulator and should not be interpreted as a benchmark of only the Gaussian particle filter. Accuracy of the other filters is similar to that of the Gaussian filter and can be found in [9].

B. Speedup

The execution time of the external filters as a function of the number of used cores has been investigated. The speedup S_p is defined as the execution time when using one core over the execution time of the filter when using p cores, i.e. $S_p = \frac{T_1}{T_p}$. Several data collection sessions have been

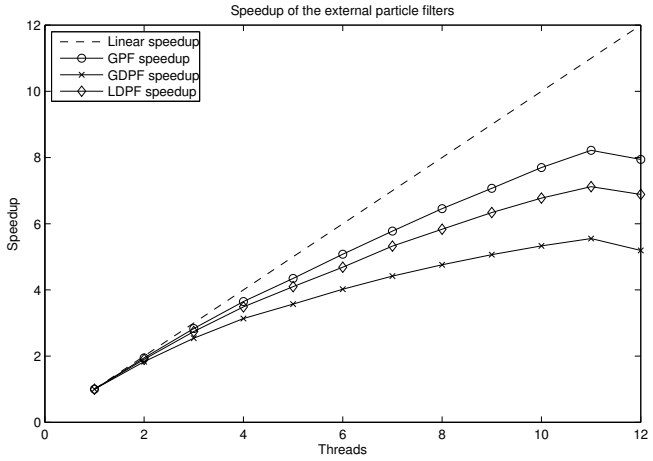

Fig. 5. Speedup of the three external particle filters used in the second phase of the tracking (Section IV)

performed and processed by each filter 50 times, varying the number of the used threads while keeping the total amount of particles fixed to 2000. The evaluated speedup is shown in Fig. 5, also depicting for reference the linear speedup as a dashed line.

V. Discussion

The platform described above can be used in an educational environment for development, testing and benchmarking of parallel algorithms, giving the tests a realism that is hard to match in a simulation. The multi-core server provides, together with the robots, a complete system able to solve, in a centralised or distributed way, many interdependent problems encountered in the part of curriculum covering sensor networks, multi-core and distributed computing.

Extending the implemented platform to a broad range of control and estimation techniques is easy, which system property has been an important design goal. The approach taken here is very suitable for education in automatic control. Robots made of Mindstorms can effectively demonstrate the challenges and actual performance of control algorithms. Many other problems than bearings-only tracking could be implemented on this kind of platform. The NXT has shown to be able to handle the computational load of modern control algorithms and still be responsive. The Mindstorms user community is large and many projects devoted to control problems ranging from inverted pendulums and non-linear estimation problems to simple PID-controllers have been successfully completed and presented on the internet.

A disadvantage of this implementation is that the network bandwidth was shown to be quite low. This does not look like a hardware problem since bluetooth has higher bandwidth than the experementally measured 3.7kB/s of maximum throughput. Some work went into improving the bandwidth, but the results were not satisfactory. To get around this problem, the preprocessing filter was used to lower the bandwidth needed, but it also caused a loss of resolution. Extending the network capabilities to include scatternets would enable the platform to work with even more advanced problems using distributed sensor networks.

The speedups of the external filters are shown in Fig. 5. The Gaussian particle filter and the locally distributed particle filter have almost no sequential parts and are expected to run with a close-to-linear speedup. The lower speedup of the globally distributed particle filter is due to the relatively larger portion of sequential code, compared to the other filters. Both normalization of the particle weights and re-sampling are done sequentially, whereas they are done in parallel in the other two filters.

Sufficient accuracy of position estimation of the sensor nodes is crucial for the tracking of the non-cooperative target. The accuracy in the second phase of the filtering is highly dependent on whether the first phase of the filtering is sufficiently accurate or not. As shown in Fig. 4, tests confirm the platform's ability to keep track of the non-cooperating target in runtime. The robots position estimate could easily be improved provided more sensors were used. Sensor fusion by adding compasses or accelerometers would improve the estimate and make online calibration possible.

References

[1] J. Takahashi, K. Sekiyama, and T. Fukuda, "Cooperative object tracking with mobile robotic sensor network," in *Distributed Autonomous Robotic Systems 8*, H. Asama, H. Kurokawa, J. Ota, and K. Sekiyama, Eds. Springer Berlin Heidelberg, 2009, pp. 51–62. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00644-9_5

[2] M. Wheeler, B. Schrick, W. Whitacre, M. Campbell, R. Rysdyk, and R. Wise, "Cooperative tracking of moving targets by a team of autonomous UAVs," in *2006 IEEE/AIAA 25th digital avionics systems conference, Vols 1- 3*, ser. Digital Avionics Systems Conference, 2006, pp. 899–907, IEEE/AIAA 25th Digital Avionics Systerms Conference, Portsmouth, OR, OCT 15-18, 2006.

[3] M. Coates, "Distributed particle filters for sensor networks," in *IPSN '04: Third international symposium on information processing in sensor networks*, 2004, pp. 99–107, 3rd International Symposium on Information Processing in Sensor Networks, Berkeley, CA, APR 26-27, 2004.

[4] I. M. Rekleitis, "A particle filer tutorial for mobile robot localization," Technical report TR-CIM-04-02, Center for Intelligent Machines, McGill Univeristy, 3480 University St., Montreal, Québec, Canada H3A 2A7, Tech. Rep., 2004.

[5] A. W. Stroupe, M. C. Martin, and T. Balch, "Distributed sensor fusion for object position estimation by multi-robot systems," vol. 2, 2001, pp. 1092–1098, cited By (since 1996): 30. [Online]. Available: www.scopus.com

[6] Lego mindstorms. http://mindstorms.lego.com/. Viewed 2011-02-04. [Online]. Available: http://mindstorms.lego.com/

[7] M. Gasperi, P. Hurbain, and P. Hurbain, *Extreme NXT: Extending the Lego Mindstorms NXT to the Next Level*, ser. Technology in Action. Apress, 2009. [Online]. Available: http://books.google.com/books?id=hxt63NxJvPEC

[8] Lejos. http://lejos.sourceforge.net/. Viewed 2011-02-04. [Online]. Available: http://lejos.sourceforge.net/

[9] F. Wahlberg, "Parallel algorithms for target tracking on multi-coreplatform with mobile lego robots," Master's thesis, 2011. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-155537

[10] F. Cristian, "Probabilistic clock synchronization," *Distributed Computing*, vol. 3, pp. 146–158, 1989, 10.1007/BF01784024. [Online]. Available: http://dx.doi.org/10.1007/BF01784024

[11] O. Rosén, A. Medvedev, and M. Ekman, "Speedup and tracking accuracy evaluation of parallel particle filter algorithms implemented on a multicore architecture," in *Control Applications (CCA), 2010 IEEE International Conference on*, 2010, pp. 440 –445.

[12] Google. Protocol buffers. http://code.google.com/p/protobuf/. [Online]. Available: http://code.google.com/p/protobuf/