A programming framework for a group of multiple mobile robots moving in a real world.

Tomoaki Yoshida[1], Akihisa Ohya[2], Shin'ichi Yuta[1]
[1] University of Tsukuba
{yos,yuta}@roboken.esys.tsukuba.ac.jp
[2] University of Tsukuba/JST
ohya@roboken.esys.tsukuba.ac.jp

Abstract

In this paper, we discuss about a programming framework for multiple mobile robot. The framework is targeted on a group of real robots operating in proximity, especially robots in formation. In such case, following major problems should be considered. First problem is about coordinate system which behaviors are described. Second is about timings of starting and ending actions.

To handle these problems, some functionalities that the framework should provides are proposed. Furthermore, an implementation of the framework is described.

1 Introduction

There has been many researches like in [1], [2], [4] referring on the field of multiple mobile robot. One of the typical tasks for multiple mobile robot is navigation task by formation of multiple mobile robots (Figure 1). To program behaviors of a group of real robots, actions of each robot in the real world need to be defined clearly and easily. Such definition can be done using position, starting timing, and ending timing of determined actions. In case of operating in real world, datum reference and timings are not matched among robots, without any matching system. In this paper, a programming framework for multiple mobile robot aiming to work in real world and its implementation is described.

In order to make this problem simpler, we focus on a case which a group of real robots operates in proximity, especially robots in a formation. For example, a group of robots deploying a train formation without any physical connection, navigates by tracking a virtual railroad. Even with this assumption, some mechanism for sharing datum reference and synchronizing actions timing is mandatory.

Figure 1: An example navigation task by a formation of mobile robot

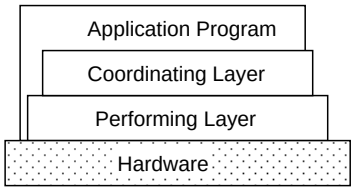

Figure 2: A layered structure of the framework core.

2 Strategy

In a specific robot, a target task can be split into two parts(Figure 2). One is a coordinating layer which communicates with other robots and coordinates cooperative behavior. Another is a performing layer which performs a motion plan given by the coordinating layer.

In the coordinating layer, communicating with robots of the group, a motion plan for achieving such given cooperative behavior for each robot is generated. A motion plan is described upon a coordinate system which has common datum reference among robots. This motion plan should not be containing any conflicts among robots at this point.

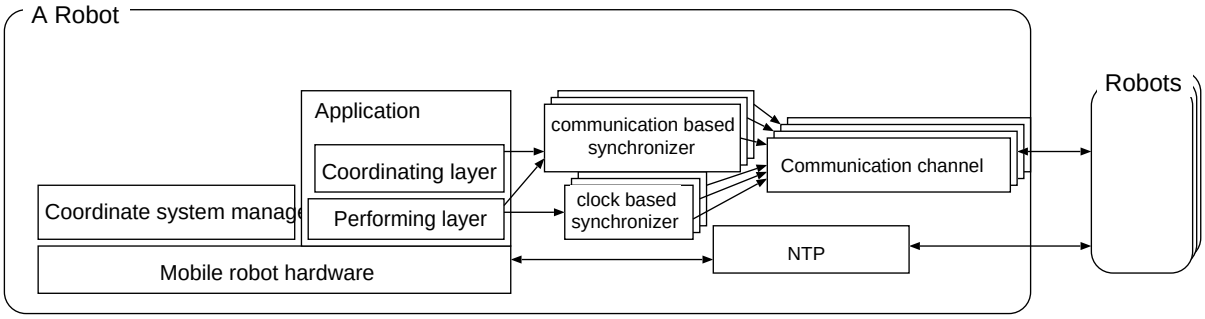

Figure 3: Software modules in the framework.

Given a motion plan, a performing layer executes such plan, which contains a set of motion commands and time codes for synchronization among robots. Referring time codes, allow the performing layer adjusts parameters of actual low level locomotion commands for motion command and to synchronize motions with other robots.

2.1 Performing Layer

A performing layer is responsible for motions of a single robot. Motions are described on a common coordinate system and triggerd by a common event source among robots to achive a cooperative behavior.

The framework provides a common coordinate system manager to help sharing a datum reference of motion description among robots. Comparing measuring results of same environment among multiple robots, it is possible to detect the coordinate system error. Once a difference of coordinate system is detected, by using a priority table given in advance, low priority robot adjust its coordinate system to match the coordinate system of higher priority robot. In this matching strategy, the highest priority robot is considered as a reference robot and all other robots try to adjust their coordinate system to match to the highest priority robot.

Meanwhile this approach does not require any information about the environment in advance, it can be applied in an unknown environment. Furthermore, temporarily placed object which is not on maps usually, such as cardboard containers can be utilized, if robots operate in proximity and measure same object almost at the same moment.

For sharing triggers of motions, the framework provides two kinds of timing synchronizers. One is a clock based synchronizer, where assuming that each robot has a real time clock synchronized with NTP, set of motions described in advance can be triggered by using clocks. Another is a communication based synchronizer which is for external event source, such as start timing supplied by human.

In a case when a performing layer detects that motions can not be realized, it notifies to coordinating layer and hopefully, it provides re-planned feasible motions.

2.2 Coordinating Layer

The coordinate layer is responsible for behaviors of the group of robots. It plans a set of motions for each robot in order to achieve a task goal, and communicates with robots in a group to resolve any possible conflicts.

The framework provides communication channels and also a voting system built on top of communication channels, for communication and negotiation with robots in a group.

3 Implementation

3.1 Robot system

We use the YAMABICO(see Figure1) mobile robot platform. Each robot is equipped with a PC runnning Linux as its main controller and communicates via IEEE802.11b wireless LAN with other robots. Functions of low level control layer such as motor servo control, position estimation by odometry, ultrasonic range finding are implemented on function modules that has a Transputer as processor on each card[5]. All other functions such as main decision making, wall detecting are implemented on a notebook PC. The framework is a set of software libraries implemented on a notebook PC.

3.2 Coordinate system manager

The coordinate system manager corrects coordinate system definition errors among robots. While it is implemented as independent thread working in background, any other software module is not necessary to treat with it once it has been activated.

Two robots named R_A and R_B, which measure position and pose of a same landmark, based on a common coordinate system, including position of landmarks(P_A, P_B) as well as pose of those given landmarks(θ_A, θ_B) (see Figure 4). If both robot's coordinate systems have same definition, then resulted measurements(P_A and P_B, θ_A and θ_B)

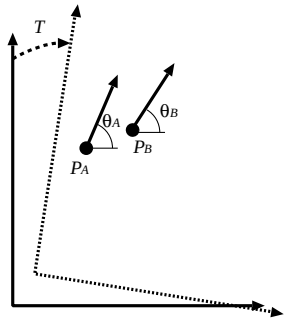

Figure 4: Coordinate system matching

are supposed to have a same values. On the other hand, if two robots have different definitions of the coordinate system, then resulted measurements difficulty can be matched. It means that small differences of measured position and pose represent a difference of each robot's coordinate system definition. Following an affine transformation T that transforms a coordinate system from R_B into R_A.

$$T = \begin{pmatrix} \cos\theta & -\sin\theta & P_A x\cos\theta + P_A y\sin\theta + P_B x \\ \sin\theta & \cos\theta & P_A y\cos\theta + P_A x\sin\theta + P_B y \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

$$R_A = TR_B$$

where

$$\theta = \theta_B - \theta_A$$

By means of this transformation T, it is possible to redefine a coordinate system of R_B and make possible two robots to share a same coordinate system as result.

This method does not assume the existence of an absolute coordinate system tied to an environment. Also, this method does not require any information of the environment in advance, robots will temporaly make usage of information of some objects in the environment, which usually does not appear on conventional maps.

When sensors on robots don't have any ability to measure 3 degrees of freedom, then unique transformation T could not be determined. In this case, a determined number of constraints can be established, which will depend on the number of calculated degree of freedom. Considering these constraints and redefining a coordinate system in order to satisfy their differences between both coordinate systems, ambiguity can be reduced.

Since there is difficulty to judge whether measured objects are identical or not, this method hardly applied to correct initial definition error. In our framework, initial definition should be done manually, and then accumulating errors as well as initial error, which are expected to be small, are corrected with this coordinate system manager.

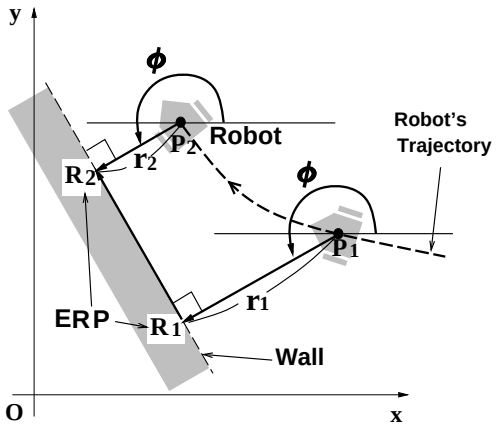

Figure 5: Configuration for the calculation of ERP. r_1, r_2 are ranged data from robot's position P_1, P_2, respectively. When these ranged data originated from the same flat wall, ERP R_1, R_2 are on the intersections of the flat wall and two perpendicular lines through P_1, P_2.

3.2.1 Landmark measurement

To demonstrate that our proposed system can be implemented just by using low cost sensors, we utilized mobile robots equipped with ultrasonic range sensors. Although ultrasonic range sensors only have the single ability to measure distances to closest objects in their sight angle. Therefore it is possible to determine the distance to the object as well as its direction when multiple measurements from different positions are matched, assuming that reference objects are flat walls. As a result, one degree of freedom of position and pose of a wall are hopefully measured.

As shown in Figure 5, let us consider that a couple of ranged data r_1, r_2 are obtained by ultrasonic sensors which are fitted on the left side of the robot when the robot was located at $P_1(x_1, y_1)$, $P_2(x_2, y_2)$, respectively. If these ranged data were originated from the same flat wall, then the reflection points on the wall should be on the intersections of the flat wall and two perpendicular lines through P_1, P_2. Because ultrasonic waves are reflected specularly over the flat wall surface. We call these points Estimated Reflection Point (ERP)[3]. Now, we name two ERP as R_1 and R_2. The vectors $\overrightarrow{P_1R_1}$ and $\overrightarrow{R_1R_2}$ are perpendicularly, then the inner product of these vectors should be 0 as shown by the following equation.

$$\overrightarrow{P_1R_1} \cdot \overrightarrow{R_1R_2} = 0 \tag{2}$$

The angle ϕ denotes a direction of the ultrasonic reflection and 0 degree is set on a direction of x-axis of the coordinate system of the robot and anti-clockwise direction is set to positive value. Using components of vectors $\overrightarrow{P_1R_1}$ and

$\overrightarrow{R_1R_2}$ are expressed as follows:

$$\overrightarrow{P_1R_1} = \begin{pmatrix} r_1 \cos\phi \\ r_1 \sin\phi \end{pmatrix} \quad (3)$$

$$\overrightarrow{R_1R_2} = \begin{pmatrix} x_2 + r_2 \cos\phi \\ y_2 + r_2 \sin\phi \end{pmatrix} - \begin{pmatrix} x_1 + r_1 \cos\phi \\ y_1 + r_2 \sin\phi \end{pmatrix} \quad (4)$$

By substituting equations (2) and (3) for equation (1), ϕ is calculated as follows.

$$\phi = \frac{\pi}{2} - \alpha \pm \arccos\left(\frac{-(r_2 - r_1)}{\sqrt{(y_2 - x_2)^2 + (x_2 - x_1)^2}} \right) \quad (5)$$

Where, α is the angle that satisfies the following relations.

$$\sin\alpha = \frac{x_2 - x_1}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}} \quad (6)$$

$$\cos\alpha = \frac{y_2 - y_1}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}} \quad (7)$$

After the calculation of the value ϕ as mentioned above, we can determine the position of the two ERP corresponding to a couple of ultrasonic range data.

Therefore, resulted ERPs are calculated with assumption that they are on a same flat wall, it is required to confirm that ERPs are really on a same flat wall. Firstly, making a group of ERPs that found sequentially and fit a line segment using least-squares method. If one of ERP in the group is originated from an object other than a same flat wall, distance between the ERP and the line will be larger, and also a vector $\overrightarrow{P_1R_1}$ and the line will not meet at right angles. In such case, the ERP group can not be considered that were originated from a same flat wall.

An ERP group that already passes the tests, can safely be considered as a part of a flat wall in the environment. Resulting ERP group represents one degree of freedom of position and pose of a flat wall.

3.2.2 Managing landmark information

At each event when a robot finds a flat wall, the robot itself notifies information that a wall has been detected to the rest of the robots. Every robot stores reported information in its database. If received information is announced by itself, then matching wall information is searched from other robots information and if received information comes from other robot, then matching wall information is searched from information announced by itself.

Old information in a database is useless, because the common coordinate system continuously and gradually accumulates odometry errors and changes by itself. Thus such information are purged from a database when they have aged enough.

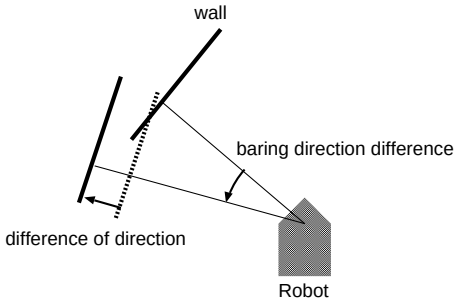

Figure 6: A pair of wall and matching operations.

3.2.3 Find matching walls

Examining a transformation for coordinate redefinition can be calculated as described in later sections, a pair of walls is determined if they are the same wall in the environment. The following conditions are examined.(see Figure 6.)

1. A baring direction difference of two walls is less than a threshold.
2. A difference of distances being from robot to each two walls is less than a threshold.
3. Two walls are intersected if a coordinate system is redefined.

3.2.4 Redefining a coordinate system

Redefining a coordinate system to match a found wall and the reference wall, results a new coordinate system, two walls could be measured at a same position and pose.

At first, rotating around a robot's current position to match a baring direction of two walls. This operation is equivalent as changing robot's direction. Then translate to direction of normal vector of wall for a difference of distances that are from robot to each two walls.

Therefore the coordinate redefining system works independenting from robot's main decision making system, when a redefinition occurs, position estimation of robot will jump non-contiguously. This asynchronous event can be a problem. In order to avoid this problem, a coordinate system redefinition events are notified to a main decision making system.

3.2.5 Priority of reference

A pair of matching walls is found on two robots simultaneously. For deciding which robot performs a coordinate system redefinition, priority of reference of each robot is assigned in advance. A priority of reference is a concept to avoid reference loop and it is represented by unique integer value mapped to each robot.

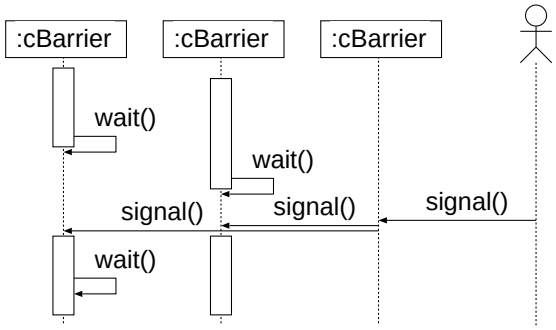

Figure 7: Communication based synchronizers for external event source.

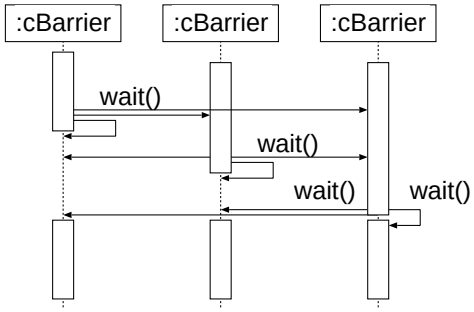

Figure 8: Communication based synchronizers for internal pre-defined condition.

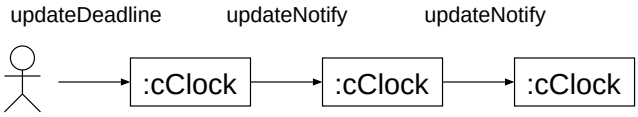

Figure 9: A chain of clock based synchronizer (re)configuring their deadline.

When a matching wall is found, a robot which has lower priority of reference performs a coordinate redefinition to match its coordinate system to higher priority robot. A lower priority robot need to redefine its coordinate system during more time than the higher priority robot. Accordingly, the robot which has highest priority of reference does not redefine its coordinate system and becomes as the reference coordinate system holder.

3.3 Timing synchronizer

Since primary target of the framework is a navigation in a formation task, tight synchronization is not required. Therefore we decided to put synchronized points sparsed in a set of motion commands.

Motions can be synchronized if starting time and ending time of each motion are clearly defined, but it is difficult to determine every timing in advance. Also some timings are decided at runtime. For example start timing of a task may be indicated by an external event source such as user command at runtime.

To resolve these timing problem, the framework provides two kind of timing synchronizers. One is a communication based synchronizer which provides functionality to synchronize external event source, and another is a clock based synchronizer which provides functionality to handle real time clock. Basically both timing synchronizers are used in performing layer.

3.3.1 Communication based

The communication based synchronizer is used to synchronize action timing which actual time of occurrence is not known in advance. When starting up a task, task programs are launched asynchronously, thus synchronization is needed to match actual starting time of the task.

Two cases of scenarios that have signaled from external event source and synchronize with pre-defined condition are considered. Figure 7 shows an event flow of external event source synchronizer. Synchronizer objects('cBarrier' shown above in Figure 7) has same ID to communicate each other. Before signaled, synchronizer objects will be blocked based on *wait()* call. When *signal()* method is called, all synchronizer objects with same ID are notified and blocked, and *wait()* calls are released. A *wait()* call on signaled object will not block. Due to the fact that a start timing of task programs can be synchronized with a signal from UI program by using this synchronizer.

Figure 8 shows an event flow of pre-defined condition synchronizer. Synchronized objects has same ID and same unblocking condition list which is a list of required member names. When a *wait()* method is called, a synchronizer notifies all other objects on the network including itself, that it entered barrier. If an unblocking condition list of notified object is satisfied, then blocked *wait()* call will be released. In contrast with external event source synchronizer, there is no explicit *signal()* call and any *wait()* call may release a blocked *wait()* call depending on the condition list. If there is no explicit event source and just need to synchronize a task execution flow, this pre-defined condition synchronizer will be used.

3.3.2 Clock based synchronizer

Once staring timing is synchronized, the next motion commands are synchronized using by real time clock and do not require communication among robots while a target behaviour itself is common among robots.

Clock based synchronizer is one of the utilities object for performing layer programming. It has one deadline per object and manages it. Unlike communication based syn-

```
mess_init();
// invoke coordinate system manager
cWallCollector::
     theWallCollector().run();
cCoordinateManager::
     theCoordinateManager().run();
// move to start position
lineup(theRobot().name);
//wait for start signal
cBarrier sync("Init");
sync.wait();
```

Figure 10: An example of initialization code.

```
cBehavior plan;
int off=thrHostConfig().priority()*30;
cMotion motion(
          cClock::relative(30),
          cMover(0+off,0,1000+off,0));
plan.push_back(motion);
plan.proceed();
cException e=plan.nextException();
```

Figure 11: An example of minimal navigation code.

chronizer, clock based synchronizer is not intended to use separately but linking multiple synchronizer object to resolve absolute time of synchronizing point(Figure 9.

The clock based synchronizer has ability to: (1)wait until deadline is reached, (2)query time for the deadline, (3)notify other refering object about update of deadline, (4)(re)set deadline from other objects-deadline and time offset.

A minimal set of motion description has one clock based synchronizer object and at least one motion command object. With a clock based synchronizer, motion command object adjust its parameter (such as velocity) to meet deadline. If it is not possible to adjust parameter in an acceptable range, motion command object notifies exception to higher layer.

4 Programming with framework

Figure 10 shows an example of the initialization code. First of all, coordinate system manager(cWallCollector and CoordinateManager) is invoked, and then move to initial position for current robot. Although this code is executed asynchronously on each robot, execution flow will be synchronized with communication based synchronizer(last line).

Once initialization has been done, behavior can be described. Figure 11 shows a minimal navigation code. *theHostConfig()* is a host specific data storage and off will set to a geometrical offset for current robot. A *motion* is a minimal motion command which indicate to run 10m in 30 seconds.

5 Summary

In this paper, a programming framework for a group of multiple mobile robots has been described. The framework is targeted on tasks which robots operates in proximity such as a navigation in robots formation, and aims to program such task in an easily way.

As a future work, we will consider about more flexible implementation of the performing layer. With current implementation, a detailed motion description is needed and is tight geometry-based. It should have more flexibility for easy programming to generate a behavior.

References

[1] Jakob Fredslund and Maja J Mataric. Huey, dewey, louie, and gui – commanding robot formations. In *Proceedings of the 2002 IEEE International Conference on Robotics & Automation*, May 2002.

[2] Aaron Khoo and Ian Douglas Horswill. An efficient coordination architecture for autonomous robot teams. In *Proceedings of the 2002 IEEE International Conference on Robotics & Automation*, May 2002.

[3] Akihisa Ohya Takashi Yamamoto, Shoichi Maeyama and Shin'ichi Yuta. An implementation of landmark-based position estimation function as an autonomous and distributed system for a mobile robot. In *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1141–1148, Oct. 1999.

[4] Richard T. Vaughan. Exploiting task regularities to transform between reference frames in robot teams. In *Proceedings of the 2002 IEEE International Conference on Robotics & Automation*, May 2002.

[5] Shin'ichi Yuta. Autonomous self-contained robot 'yamabico' and its controller architecture. In *Third Australia National Conference on Robotics*, June 1990.