Preliminary Development of Subsumption Architecture Control for Automated Guided Vehicle

Prianggada I. Tanaya[a,] *, Saras Safitri[b]
[a]Department of Industrial & Systems Engineering, Faculty of Engineering, International University Liaison Indonesia
Eco-Campus, GOP6, The Breeze, Tangerang 15345, Indonesia
[b]Department of Mechatronics, Faculty of Engineering & Information Technology, Swiss German University
EduTown BSDCity, Tangerang 15339, Indonesia
*E-mail: prianggada.itanaya@iuli.ac.id, safitri.saras@gmail.com

Abstract

In manufacturing automation setting, responses to input is considered mission critical. Information from sensory input propagate hierarchically before decision making executed. Automated Guided Vehicle (AGV) is an automated component as material handling of manufacturing systems. To increase responses upon sensory feedback, a flat architecture called subsumption is introduced. It is based on parallel system. Sensory input is directly connected through modules in the control system, upon the sensory information processing finalized, a decision making selected and send an appropriate command to the actuators. In this article, this control architecture will be designed and implemented to an AGV. Commands are designed based on Object-Oriented technology. The commands are arranged in subsumption, where a command higher subsumed other command of its lower level. GPFO (Greater Priority First Out) technique is implemented for executing the commands by using multi-threading methodology. Experimentation is performed to have the characteristics of commands being executed. This work introduce our effort to design an operating system for an AGV.

Keywords: *AGV; subsumption; control architecture; GPFO (Greater Priority First Out); object-oriented programming*

Abstrak

Dalam pengaturan otomasi manufaktur, tanggapan terhadap input dianggap misi yang penting. Informasi dari masukan sensor merambat secara hierarkis sebelum pengambilan keputusan dilakukan. *Automated Guided Vehicle* (AGV) adalah komponen otomasi sebagai piranti penanganan material pada sistem manufaktur. Untuk meningkatkan tanggapan atas umpan balik sensor, diperkenalkan sebuah arsitektur pipih yang disebut *subsumption*. Hal ini didasarkan pada sistem paralel. Masukan sensorik secara langsung terhubung melalui modul dalam sistem kontrol, setelah pemrosesan informasi sensorik diselesaikan, pengambilan keputusan dipilih dan selanjutnya perintah yang sesuai dikirimkan ke aktuator. Dalam artikel ini, sebuah arsitektur kendali akan dirancang dan diterapkan pada AGV. Perintah dirancang berdasarkan teknologi Berorientasi Objek. Perintah-perintah disusun dalam *subsumption*, di mana perintah yang lebih tinggi disertakan perintah lain dari tingkat yang lebih rendah. Teknik GPFO (*Greater Priority First Out*) diterapkan untuk mengeksekusi perintah dengan menggunakan metodologi *multi-threading*. Eksperimen dilakukan untuk memiliki karakteristik perintah yang dieksekusi. Karya ini memperkenalkan upaya kami untuk merancang sistem operasi bagi AGV

Kata kunci: AGV; subsumption; arsitektur kendali; GPFO, pemprograman berorientasi objek

1. Introduction

The development of AGV has been deployed in many aspects and the applications at industries have increased the efficiency and productivity, especially at automated manufacturing system. There are many new developments that can be implemented on AGV system. One development is on mobile robot architecture. To control an AGV, an architecture of control is developed to synchronize all systems' components. A control architecture is a framework to control actuators, sensors, electro-mechanical system, and system software in term of modules. Subsumption architecture was introduced by Brooks [1]. The subsumption architecture is a layered architecture. The layer is organized into horizontal division (Figure 1). The division is used to operate the robot at its levels of competence. The higher level subsume the lower levels, while the lower level work dependently. Each layer corresponding to the level of behavior. With Subsumption architecture, the AGV could react quickly in dynamic or unpredictable environments. The identification of the architecture is following:

- multiple goals: the system works based on priority goal. Higher priority goal implemented to avoid conflict with lower priority goals,
- multiple sensors: parallel sensor reading at multilevel of competence, that will be forward to appropriate module to react,
- addivity: sufficient power supply throughout the system components, and
- robustness: capable to adapt if any sensor fail to react and forward the information.

Figure 1 shows a parallel horizontal layer of behavior that composed by the architecture for a mobile robot such as AGV. Sensors are connected throughout the levels, and accordingly the actuators received command from those level based on certain logical reason to react actively.

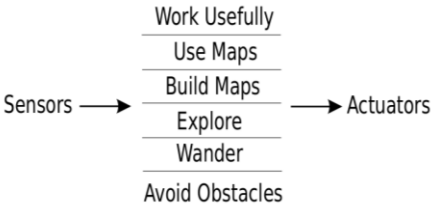

Figure 1. Subsumption Architecture [1]

Simpson et.al. [2] explored subsumption architecture for general purpose robotics hardware to provide clean and robust development effort. Occam-pi is used as development tool. This work mapped the robotics architecture with Occam-pi and shown the behavioral characteristics such as inhibition, suppression and subsumption of robot tasks to manipulate the environment. Constructing or designing behavior is one particular concern of subsumption architecture. Rosenbalt & Pyton [3] effort is to avoid rigid decompostion of commands into more flexible commands structure to construct behavior but still preserving the level of competence. A concept of NIO (Non-Interruptable Operation) are introduced on machine tool controller [10]. This NIO and/or IO are used to protect whether an instruction is interrupt-able or not. Using NIO guarantee that the instruction is completely executed, it cannot be interrupted. This type of instruction is necessary for low level commands. The study on planning, execution, and monitoring (PEM) method is implemented on industrial forklift. The work of Heikkilä & Roning on PEM-modeling [8], inspired the design and development of our AGV.

This following paper introduced our effort to design and implement subsumption architecture for controlling AGV [8]. Overview of our AGVs platform is presented in the next paragraphs, and it follows with the design of control software, implementation to the AGV, and the discussion of commands that construct the preliminary behavior. The execution of commands is presented.

2. Automated Guided Vehicle : design of instructions and implementation

Research on AGV has been performed since 2012 in our group. It focuses on obstacle avoidance and localization, maneuverability, path planning [4,5,6], and prototype of subsumption architecture for AGV [9]. The electro-mechanics platform is shown at Figure 2. The system consists of among other, Microprocessor boards (Arduino ATMega 2560), DC motors, encoders, and PC's i5-3540 system. The whole dimension is 630x600x550 mm, and the weight is 115 kg.

Figure 2: One of The AGV platform [9]

2.1 Design of Instructions

There are 3 level of instructions, namely, the atomic command, sub-task and task. Hierarchically, the atomic-command is the lowest level command, and the task is the highest. The atomic-command is designed to facilitate interaction with basic functions of actuators and sensors. A sub-task encapsulates a set of atomic-commands and the logic required, and similarly, a task encapsulates a set of sub-tasks with a logic higher to perform the behavior of the system. Each of instruction, is in the form of object. For sub-tasks' and task's object, planning, execution and monitoring (PEM) modules are developed.

The atomic commands have their own planner, these command instructions interact directly with ready to execute routine at Arduino board. The atomic commands are designed as NIO (Non-Interruptable Operation), while the sub-taks and task as IO (Interruptible Operation). Next paragraph discusses these further. Figure 3 shows process block diagram of designed subsumption architecture.

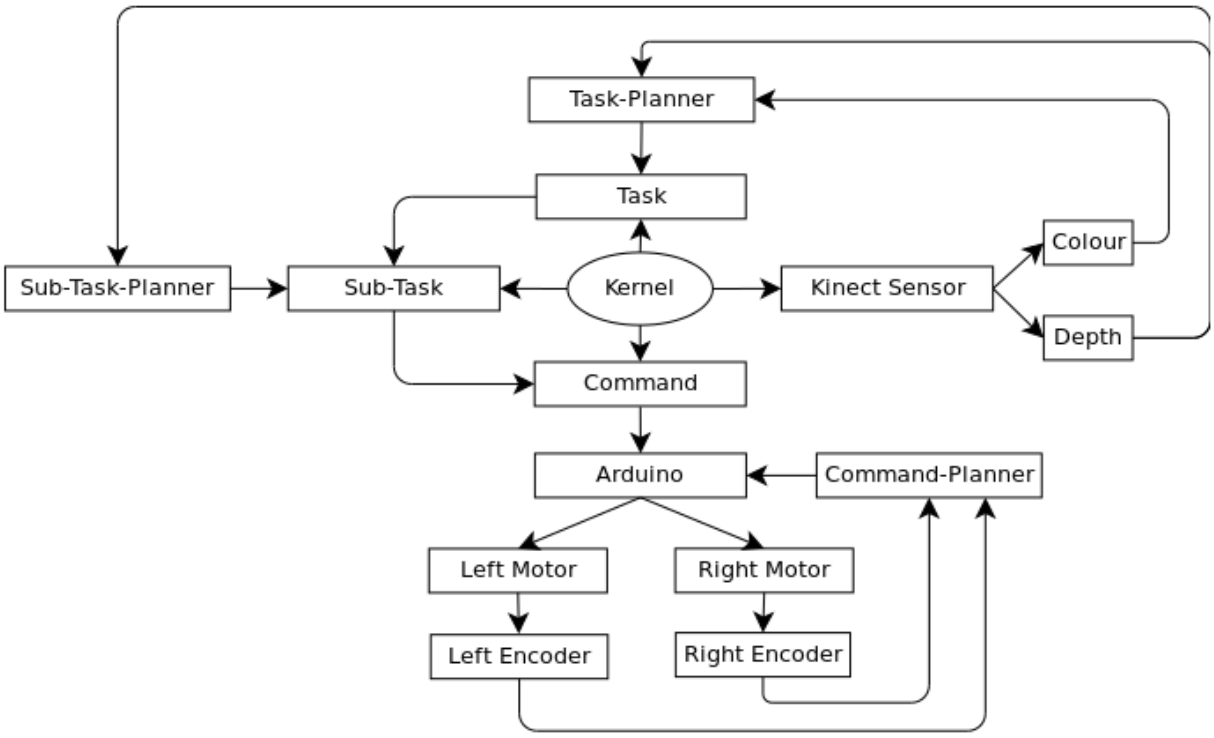

Figure 3: Process block diagram of designed subsumption architecture [9]

2.2 Implementation of Layers of Instructions

Object-Oriented Programming (OOP) method is used for the software application development. They are two level of control command, namely, located at the PC motherboard and Arduino. The Arduino boards interact directly to the actuators and sensors. The implementation of the subsumption architecture is realized, among others, using multi-threading, producer & consumer concept, signals and slots mechanism (event-based), and GPFO (Greater Priority First Out) method. The programming languages are GNU C++ on the PC side and C at the Arduino side.

The highest behavior of the AGV is described at the task level. The task is which decomposed further into the instruction lower, and further into the instruction that is understood by the actuators and sensors in term of atomic-commands. Figure 4 shows the decomposition of a task into subtasks, and a subtask into atomic commands.

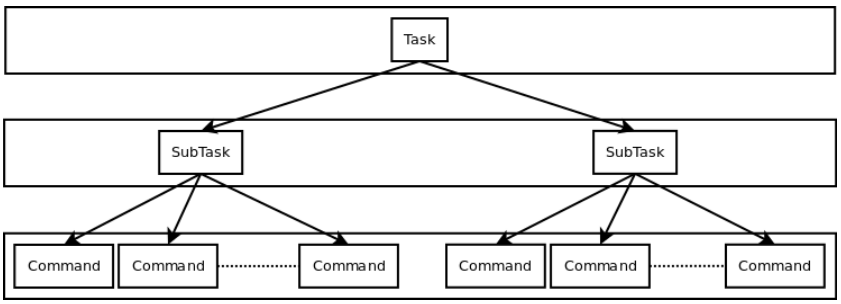

Figure 4: The decomposition of instruction: from a task to subtasks, and from a subtask to atomic commands. Those horizontak box shows the GPFO buffer of each appropriate instructions [10]

Each level of instruction is served by two threads, namely for reading and writing instruction from/into the GPFO buffer. Each tread has separate flow control which shared or communicated the data.

Producers & consumers concept is one common threading scenario. A thread which named producer have a role to generate the information and push them into an appropriate buffer. Other thread, consumer will take the information form the buffer. Producer & consumer work in parallel and both are synchronized (using mutex and semaphore) [7].

Figure 5 shows the producer & consumer part of creating the atomic command object (e.g. `MiFwdCmd`) and later this object will be consumed. It shows the classes-view using UML (Unified Modeling Language) of OOP.

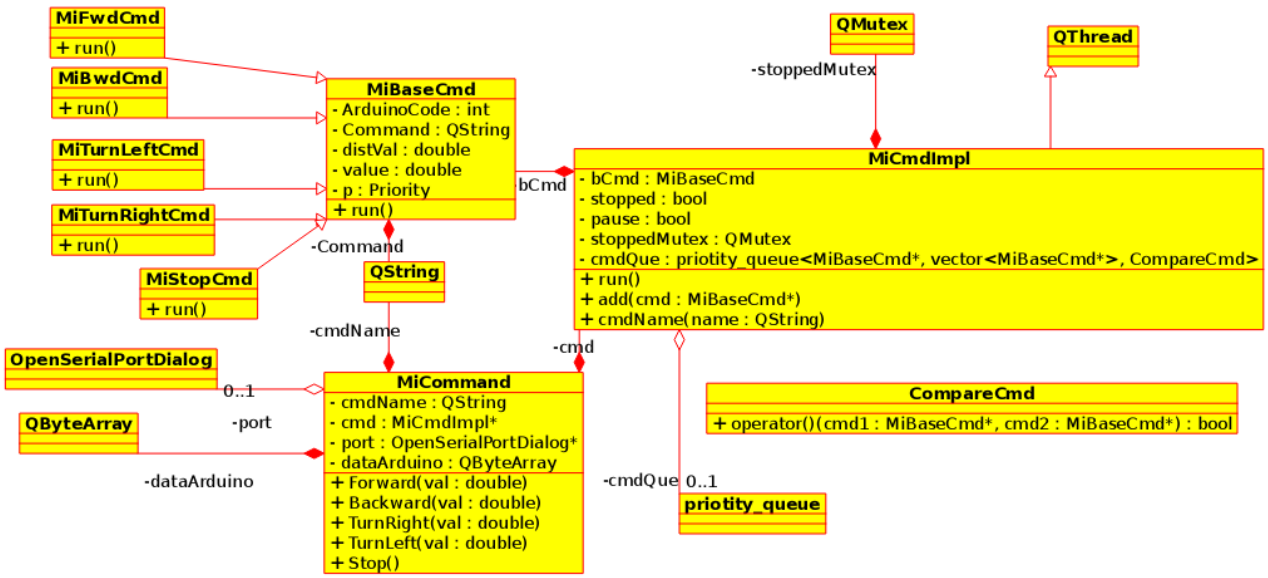

Figure 5: An UML model for creating atomic-commands through `MiCommand` and consuming them at `MiCmdImp`

The object of `MiFwdCmd` is produced by invoking `Forward` command of class `MiCommand`. `Forward` command, create an object of `MiFwdCmd`. This object is stored into a GPFO buffer, called `priority_queue`. This buffer is a shared memory buffer of type object with object's priority. The consumer part is taken the most left object after it is sorted follow GPFO priority rule. The atomic-command `MiFwdCmd` is executed into the Arduino board.

When the atomic-command at Figure 5 (such as `MiFwdCmd`, `MiStopCmd`, `MiTurnRightCmd`) objects are produced, immediately, it is stored at `cmdQue`. This `cmdQue` is an attribute of the class `MiCmdImpl` and it has a type of `priority_queue`. When atomic-command has been stored into the buffer, the producer `MiCommand` object has no control anymore. The operation `run()` of the class `MiCmdImpl` consume atomic-command objects. An ownership process of producing and consuming atomic-commands to and/or from the buffer is guarded by a `QMutex` type object. Similar to produce and consume of atomic-command objects are also implemented to subtask and task objects.

As seen at Figure 4, the decomposed instruction is structured into 3 levels. The level of task, subtask, and atomic-command. Each of those levels have planning, execution and monitoring, in term of modules, and it is known as PEM-module. Planning module at sub-task level, for example, is responsible to arrange the sequence of atomic-commands to be created. The atomic-command sequence is produced with the same default priority to facilitate sequential execution method. The execution module responsible to execute the most left atomic object first.

At Figure 6, shows the sequence of execution at task and sub-task level indicate by the numbers. A planner module (P), an execution module (E), and a monitoring module (M) are shown at every level. The P module store instruction object at any time int the priority buffer (sequence 1). The P module release the ownership of buffer to the E module (2). An instruction object that is stored is immediately checked its priority by the E module (3). The highest priority will shuffle by iteself to the most left for execution. The E module invoke the buffer, take the object and pass it to be decomposed into sub-task instruction (see Figure 4 and sequence 5 at Figure 6). The M-module pooling the buffer for incoming messages (6), and read (7) if any progress, and in turn at send the progress to E-module (8) and P-module (9). Similar type of PEM-module execution is also performed between sub-task level and atomic command level.

The P-module at any time can add new instructions at the most left sequence, and by its priority property, this instruction will be reshuffle according to the GPFO rule. Following paragraphs describe instructions at each level above.

2.3 Command Level

The Command level is a level to generate the movements of the AGV. In this level, the AGV has to reach the distance in centimeters (cm) which has been given by the user or higher-level instruction. The Command level is divided into forward, backward, turn left, turn right, and stop movements. The input is the desired distance for forward and backward movements or angle for turn left and turn right movements. The process of the Command level is separated into two boards, PC motherboard and Arduino micro-controller board. The process in the motherboard is used

to receive input. The main process of the subsumption architecture is in this board. The Arduino is used to execute the Command and sent to the DC-motor. The movement of the AGV is controlled from the Arduino board. The input is sent to Arduino through a Serial Port. In Arduino, the DC-motors are executed using an H-Bridge and the pulse change from the Rotary encoder is read.

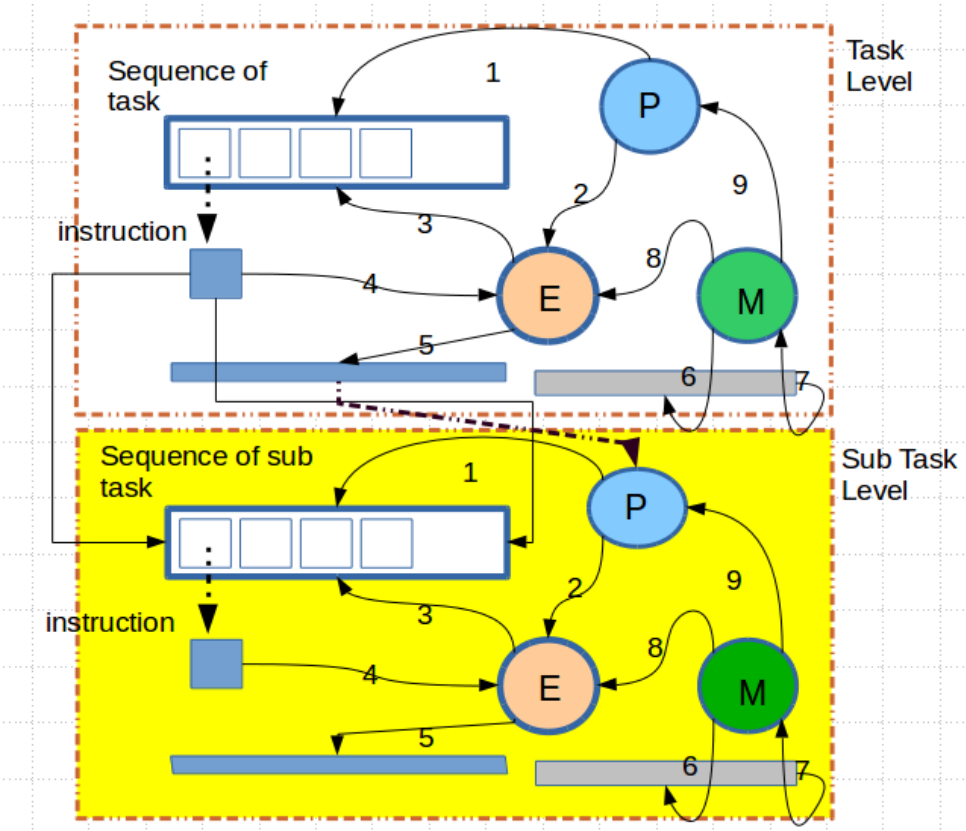

Figure 6: Task and sub-task levels, the sequence of execution of P, E, and M modules, invoking priority buffer of each level. The numbering shows the sequence of execution inter- and intra-level.

2.4 Sub-Task Level

Sub-Task generate the paths from the original point to the destination point. These points are in Cartesian coordinates which have an X-axis and Y-axis. The Sub-Task level is divided into move, rotate, and avoid programs. Unlike the Command program which programmed in the motherboard and the Arduino board. The Sub-Task level is programmed only at the motherboard. For move and rotate programs, the Sub-Task level has 3 active threads which are main thread, Sub-Task thread, and Command thread. However, for avoid instruction, additional thread is provided, namely, the thread for monitoring Kinect sensor update.

2.5 Task Level

The Task level is the highest level which can generate several Sub-Tasks. Task level is the highest level that create a behavior of the AGV to find an object. The object is detected using RGB camera from Kinect™ sensor. The position of the object is sent to sub-task level.

3. Result and Discussion : AGV Control with Subsumption Architecture

A language is developed internally at our group to operate the AGV's with trivial task. The focus of this language is to use at sub-task level and task level. Furthermore, some setting such as open serial port, open Kinect window, set the initial robot position, and close all window and connection are available. The description about the tasks for this robot language can be seen in Table 1.

Table 1: Description of Simple Atomic Command

Atomic Command	Description
`openPort<port name>`	To open serial port of Arduino
`Close`	To close serial port and Kinect window
`set<x><y>`	To set robot position at <x> and <y> in reference coordinate
`move`	To move the robot to point <x> <y>
`rotate<r><t>`	To rotate the robot with radius <r> in cm, and angle <t> in degree
`findColor<type_of_color>`	To find an object with specific color (e.g. green yello, red)

Following paragraphs elaborate some of the atomic commands of Table 1.

1. `set <x> <y>`
   The initial position of the AGV must be formerly declared using `set <x> <y>`. Point <x> <y> is the position of the AGV in reference frame. The X-axis and Y-axis of the reference frame are parallel with the X-axis and Y-axis of the AGV. Figure 7 and Figure 8 show the flow-chart and the position of the robot.

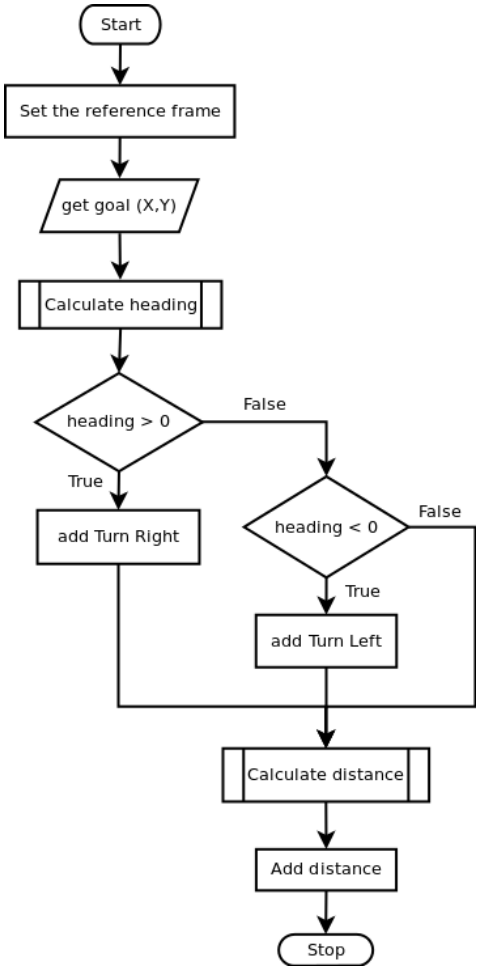

Figure 7: *`set` and `move` Flow-Chart*

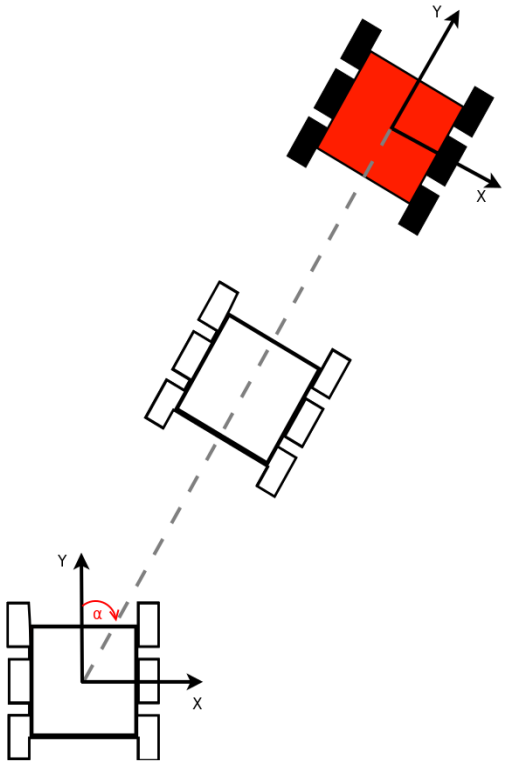

Figure 8: *`set` and `move` Path*

2. `move <x> <y>`
   Simple task of `move <x> <y>` is an instruction to create a path from initial position which to the goal point (x,y). To reach the goal, the AGV has to turn for a certain angle and then move straight for a certain distance. The AGV is not allow to move backward as there is no sensor on the backside of the robot. Figure 7 and Figure 8 show a move flow-chart and move path. In higher hierarchy, a sub-task can be composed of `move`

<x> <y> and activating Kinect sensor to construct avoidance function. Figure 9 shows path for avoiding an obstacle. The minimum distance between an obstacle and the AGV is 60 cm within a straight path plan.

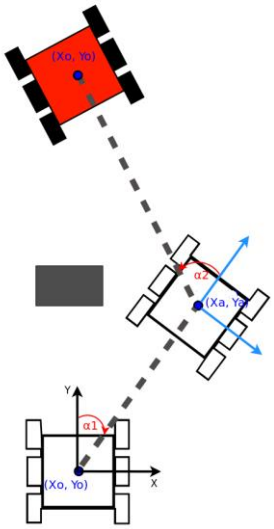

Figure 10: Avoiding an obstacle within its path

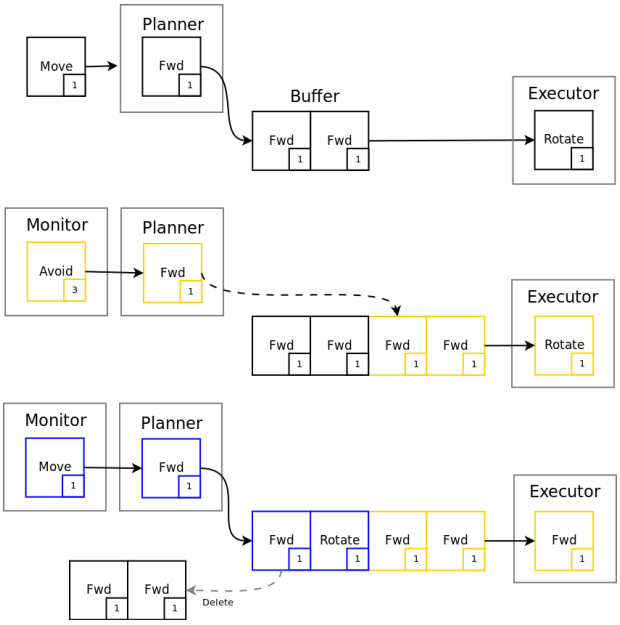

Figure 9: Execution command processes in avoiding an obstacle. Showing PEM-module : Planner, Executor, and Monitoring role. Showing their priority level at each command

Figure 10 (and also refer to Figure 6) shows avoiding obstacle processing involving the PEM-module and their buffer. Initially the AGV is executing `move` forward. `move` is in its buffer, then executed by the E-module (executor). Within its path of movement, M-module scans an obstacle within its sensing distance (60 cm). Sensing the obstacle performed by Kinect sensor. From the straight path to avoid the obstacle, the AGV placing several commands, namely, `rotate`, `move`, `rotate`, `move`, and `rotate`, to keep continue on previous path planned before encountering an obstacle. The commands rotate, `move`, `rotate`, `move`, and `rotate` are inserted into the buffer, as part of re-planning task as performed by the Planner (P-module).

At first, the sub-task processes `move`. After the first has been executed at the execution module (E-module) of the PEM-module, the monitoring module (M-module) is checking the environment using Kinect. If the AGV encounter an obstacle in its path, an avoid task will be re-planned in P-module, added into the execution buffer and follow GPFO method, it will be executed by the E-module. The P-, E-, and M-modules are threads in programming method. They are synchorized using mutexes and semaphores [7].

3. GPFO (Greater Priority First Out) Example: obstacle avoidance
   The avoid instruction process can be seen at Figure 10. It shows the execution mechanism of GPFO buffer. The first process, the sub-task processes the move instruction. After the first has been executed.

4. `rotate <r> <t>`
   The planner of the Sub-Task level generates combination of turn left or turn right movement and forward movement which is shown in Figure 11. The goal of this simple task, the AGV have to move along a circle path with radius 'r' in cm, and for an angle 't' degree. To perform the command, the following equation are needed to provide above movement.

$$L_c = \frac{2\pi R\theta}{360} \quad \text{..... Eq.1}$$

$$P = \frac{L_c}{60} \quad \text{..... Eq.2}$$

$$\alpha = \frac{\theta}{60} \quad \text{..... Eq.3}$$

$$L = \frac{L_c}{P} \quad \text{..... Eq.4}$$

where:

L : discrete forward movement distance in cm

L_c : circumference length of circular path in cm

R : radius of circular path in cm

θ : circular path angle in degree

α : turning angle from previous linear discrete path to the next linear path in cm

P : number of turning point, discrete linear path from circular path for robot movement

Rotation movement of above path is performed. The radius of rotation is 100 cm, the angle of rotation is 90º. Due to robot moment of inertia is quite high, the turning angle α cannot be less than 15º. Table 2 shows the execution time results, unit is milli-seconds (ms).

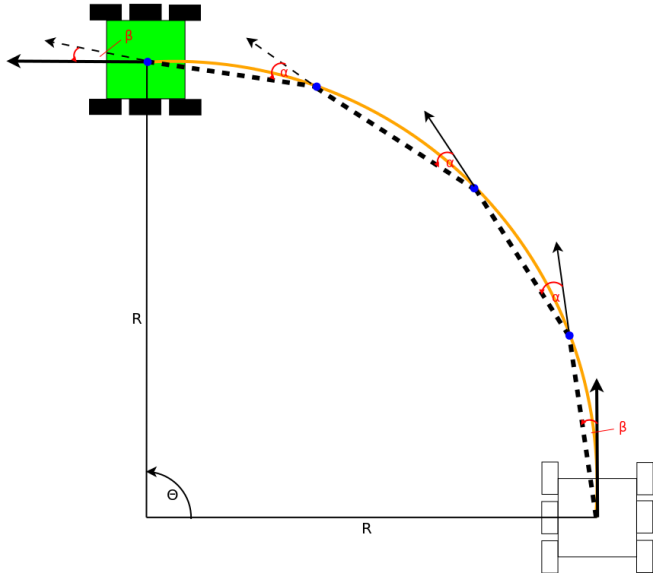

Table 2: Execution Time (ms) for sequece of turn left and move forward – piecewise circular movement

No	Turn Left (ms)	Forward (ms)	Turn Left(ms)	Forward (ms)
1	1113	3985	1000	2990
2	1205	3897	1000	2998
3	1508	3591	1000	2996
4	1125	3970	1004	3000
5	1113	3989	998	3000
6	1225	3900	995	3005
7	1100	3999	1000	3001
8	1125	3974	1000	3003
9	1113	3989	999	3024
10	1113	3985	1000	2990
Average	1174	3927	999.6	3000

Figure 11 shows consecutive turning points movement. This turning movement is performed piecewise, constructed from rotation to the left on a point (start point), and follow with moving forward to another point ahead (end point). During the experimentation, smoothly moving in circular path cannot be done without this piecewise turning movement. This might be caused by high inertia of maneuvering of the robot itself. The points on the curvature of circular path is resulted from interpolation.

Table 2 shows the execution time of turning movement (2 rotation experimentation, each has 10 points of interpolation). The measuring of execution time in milli-seconds is done through programming, read the time (CPU time) when it starts to rotate to the left (figure 11) until stop, read the time again, start to move forward in linear path, read again the time. From those CPU time consecutive reading, the time of turn-left and forward movement is calculated, by subtracting the end reading of time with the start reading of time for turn-left, and similarly for move forward. On average, the execution of time above is within 3.5 seconds.

4. Conclusion

The subsumption architecture is a particular type of mobile robot architecture system software which is divided (layered) the instruction into several level/layer. Each behavior of an instruction subsumed many behaviors of instructions in more atomic level. The command level is a level to control and monitor the actuators (DC-motor) through instructions. The instruction is forward, backward, turn left and turn right movement which have to reach a desired distance or angle. The subtask instruction level is to create the trajectory with combination of movement from the command level. There are 2 types of trajectories, namely, linear (move) and circular (rotate). The planner module generates the position of the AGV using kinematics of mobile robot and decomposed it into the command level movement. The monitoring module checks the environment and it sends avoid instruction, if an obstacle has been detected. Interpolation of circular path movement is performed. Piecewise circular movement path that composed from rotation about a point and forward movement. The task level is the higher level of instruction. This instruction level subsumed the lower level. Executing those consecutive and decomposed instruction require further synchronization using thread, mutex and semaphore techniques. As our recommendation are: Additional sensors are needed to reduce the error result in mechanical aspect and observe more obstacles in completely environment, and the AGV have many types of movement and trajectory. The planner module of each layer needs to be improved. A sort of robot language is prospective to be introduced.

References

[1] Brooks, R. A. 1986, "Robust Layered Control System for A Mobile Robot", *IEEE Journal of Robotics and Automation: RA-2, 14-23.*

[2] Simpson, et.al., 2006, "Mobile Robot Architecture – the subsumption architecture and occam-¶", Communicating Process Architecture, Peter Welch, Jon Kerridge, and Fred Barnes (es.), IOS Press: 225-236

[3] Rosenblatt, J.K, and Payton, D.W, 1989, "A Fine-Grained Alternative to the Subsumption Architecture for Mobile Robot Control", Proceeding of the IEEE International Joint Conference on Neural Networks, IEEE Press: 317-324

[4] Yohannes, A, 2012, "Designing and Constructing and Autonomous Mobile Robot with Stereovision System for Obstacle Avoidance and Localization", Bachelor Thesis, Department of Mechatronic, Faculty of Engineering, Swiss German University, Tangerang, Indonesia

[5] Tewira, W, 2013, "Development of Path Planning Based on Lagrange Polynomial Method for Dynamic Obstacle Avoidance of Takeover and Crossroad Maneuver Using Kinect™ Sensor of a Mobile Robot", Bachelor Thesis, Department of Mechatronic, Faculty of Engineering, Swiss German University, Tangerang, Indonesia

[6] Tjiu, W, 2013, "Development of Execution and Monitoring Architecture Modules for An Autonomous Human Follower Transporter Robot", Bachelor Thesis, Department of Mechatronic, Faculty of Engineering, Swiss German University, Tangerang, Indonesia

[7] Posch, M., 2017, "*Mastering C++ Multithreading*", Packt Publishing, Birmingham UK

[8] Heikkila, T, and Roning, J, "PEM-Modeling: a framework for designing intelligent robot control", Journal of Robotics and Mechatronics, Vol.4, No. 5: 437-444

[9] Safitri, S., 2014, "Designing and Implementing Subsumption Architecture of Autonomous Guided Vehicle", Bachelor Thesis, Department of Mechatronic, Faculty of Engineering, Swiss German University, Tangerang, Indonesia

[10] Tanaya, P. I, 1999, "A Task Driven Prototype CNC Machine Tool Controller Holon", Doctoral Thesis, Dept. Werktuigkunde, Faculteit Toegepaste Wetenschappen, Katholieke Universiteit Leuven, Leuven, Belgie