Petri Nets and Dual-Handed Assembly Robots

FRANS VAINIO, MIKA JOHNSSON[1)], TIMO KNUUTILA, OLLI S. NEVALAINEN
Department of Information Technology and TUCS
University of Turku
20014 Turku
FINLAND
[1)]Valor Computerized Systems (Finland) Oy
Ruukinkatu2, 20450 Turku
FINLAND

Abstract: - A method to detect and manage collision situations of the concurrent operations made by two robot arms in a shared work space is presented. Two different strategies, collision avoidance and in-turns operation are considered. The operation of a dual-armed assembly robot is modeled using Petri nets. The model is then tested by the means of an animated virtual assembly system. By observing the operation of the virtual robot, the observer can verify that the robot arms work correctly and that the possible deadlock situations are avoided successfully. Finally, the performance of the collision avoidance method and that of the in-turn operation mode are compared.

Keywords: - Assembly robots, Petri nets, modeling, motion control, collision avoidance, simulation

1 Introduction

Robotic assembling is used widely in today's highly automated production lines. In these environments, the cost of production, quality of products and operational flexibility are the key factors having an influence on the joint economy of the production. Improving the assembly performance is one possible method when aiming at reduced production costs. The kind of assembly robots is tightly tied to the needs of the production process and universal robots are becoming common due to their flexibility in situations where the lifecycle of products is short. Electronics industry is one of the most emerging fields utilizing automated production lines. The machines on the lines have been traditionally specialized to rather restricted tasks, like inserting of electronic components of certain size and feeder type. A recent trend has been the increased popularity of general purpose precision robots using one or two arms and allowing several possible packaging types for the electronic components. The idea behind their use seems to be their flexibility especially in high-mix-low-volume production.

In cases where the two robot arms can physically access the same area or space, the topic of collision avoidance has to be discussed and solved. The measures for collision avoidance apparently influence the performance of the robot system significantly. The prediction of potential collisions is difficult in general cases when the two arms are used independently from each other in a fully concurrent manner. There are numerous articles and research results proving that common variants of the problem are so complex that mathematical solutions are hard to find or are not yet known [4, 5]. Because of this, heuristics or other approximate methods are needed. In the present work we apply *simulation* to evaluate the different heuristic ideas. Hereby, a discrete event modeling method called *Petri net* [7, 2] will be used and demonstrated. In particular, the operation of a dual-handed robotic systems is modeled, simulated and visualized. The performance of the systems is then evaluated when assuming two different modes of operation:

1) *concurrent* operation with collision detection and avoidance and
2) *in-turn* operation of the same robot making the same assembly task.

In addition to the many basic questions concerning the overall performance of different robot design alternatives, we have an acute practical problem of controlling a dual-handed assembly robot equipped with a real time collision avoidance system. In order to reach a favorable division of the component placements for each arm, we need a simulator for the robot. The present study is a first step towards implementing this kind of generic software.

Actions of the robot are visualized by using animation on the computer screen. The animated model is intended to verify the functioning of the Petri model to a human observer. The actual Petri net execution and its visualization are made using a simple

scripting language, which interprets the human made Petri net model into a computer understandable form.

The plan of the rest of the paper is as follows. In section 2, we shortly describe some known simulation models of dual-handed assembly robots. We restrict ourselves to the use of Petri nets only. In section 3, we present a more advanced Petri net model having an autonomous in-fly collision avoidance system instead of the mutual exclusion based systems described in section 2. In section 4, an animated model of a virtual dual-handed assembly robot is presented. Further, four different assembly strategies to be simulated are described and some results of the simulation with these strategies are presented. In section 5, some conclusions over this work are drawn.

2 Previous work

There seems to be only few models for dual-handed assembly robots using Petri nets in open literature [1, 2]. The robotic assembly system modeled by Zhou and Leu [2] has two arms. The robot repeats a cycle of picking-moving-inserting-moving operations by both of its hands. However, since the arms operate on the same PCB and obtain components from the same feeder area, avoidance of arm collisions has to be considered. Collision avoidance is achieved by the mutual exclusion technique implemented using tokens (as semaphores) in the Petri model. Thus, when one arm is operating above the PCB area, the other is prohibited from moving into the same area. Because the arms use a common feeder, collision may also occur in the feeder area. For this reason mutual exclusion is applied for this area, too.

Originally the Petri nets consist only of the four original primitives presented by Carl A. Petri. These include two types of nodes, *places* (denoted by circles) and *transitions* (denoted by bars), edges called *arcs* (denoted by edges and arrow heads on the edges indicating the direction) and *tokens* (denoted by black dots). A Petri net is a bipartite directed graph, so an edge can connect only two nodes that belong to different types. Starting (or triggering) a Petri net transition is called by convention *firing*. A transition must be *enabled* before it can fire. To be enabled means that all input places of a transition contain one token. Effects of firing are: all enabling tokens are destroyed and, after selected transition times, new tokens are created in output places by the multiplicity of the output arcs. The initial state of a Petri net model is denoted by placing tokens on the initial places. This is called the *initial marking*.

By convention, when bars are used, it is supposed that all transitions will take place in an infinitely short time. In the real world, however, they last certain time, which can be expressed by a timed net and the transitions are then denoted by rectangles. Operations like moving from the feeder to PCB are influenced by the position of the feeder and the destination of component insertion on the PCB. However, because all the movements are between the feeder and PCB areas, an average movement time can be calculated. The variation in time of the different movements can then be represented as a kind of noise. On this coarse modeling level, the operation times are modeled as a fixed time added with a small variation drawn from a negative exponential distribution.

3 Modeling a dual-handed assembly robot

We next specify a virtual robot having concurrent operation of two robot arms and study a Petri net model capable of simulating and controlling the virtual robot or even real robots. The virtual robot with two hands is schematized in Fig. 1. On the top of the figure is the feeder bank having four feeders. The unpopulated printed circuit board (PCB) is fixed in the middle of the working table. When all placements have been done, the PCB is ejected, moved to the left and a new unpopulated PCB is fed from the right of the robot. A feeder bank is located over the working table and the electronic components are stored in the slots of the feeder bank. The two arms operate independently in cycles of component pickup-and-placement operations. The arms can be moved in the y-direction and reached out in the x-direction by independent (total of four) virtual stepper motors.

3.1 Petri net model

When the two arms are used concurrently to rise the performance of the placement operations, one must face the risk for collisions, especially if shared resources or work spaces are in use. The two arms form moving obstacles to each other. To avoid collisions and robot breakdowns, collision avoidance must be arranged. Solving a general collision avoidance problem is difficult and costly [4, 5]. Therefore, in many cases, the operations are restricted to be not fully concurrent, which changes the operation mode more or less to "in turn" type. This means that there are waiting time periods when one arm waits until the other arm has completed its task and is away from the critical space. However, the waiting reduces the overall performance of the robot. Fully concurrent operation without waiting times would be a better solution. In Fig. 3.1, a Petri net

model of a dual-handed assembly robot having fully concurrent operation is shown.

In order to be able to model the operation control of the dual-handed assembly robot described above, we need three extensions to the common Petri nets [9]. First, *inhibitor arcs* are used to disable transitions in certain situations. Inhibitor arcs are denoted as arcs with circular heads instead of arrow heads. An inhibitor arc can join only places to transitions. If there is a token in the place having an inhibitor arc pointing to a transition, the transition is disabled as long as the token is there. Secondly, in order to avoid the Petri net model to become all too complex, an abstraction mechanism, *hierarchical structuring*, is used. The hierarchy construct is called a *subnet*. A subnet is an aggregate of a number of places, transitions, arcs and other subnets. Subnets make constructing, reviewing and modifying of large models easier. Third, we show explicitly the *connections to outside* of the control logic. Hollow arrows point places, where tokens are coming from the outside of the Petri net, e.g. from a computer manipulating the simulation data. Petri nets are not commonly used to make arithmetic operations, because they are not capable to read or write physical data files. Therefore, a computer is needed to run the actual Petri net and to perform file and arithmetic operations.

The Petri net model for a two-handed assembly robot in Fig 1 consists of three main functional parts:

1) the upper part of the model forms the PCB feeding and assembly controller. The initial marking for this functional part of the net is one token in the place *OK_TO_ASSEMBLY*. During the robot's assembly operations nothing happens here until the *END_ OF_ FILE* (EOF) of the list of components is encountered, then a token in the place *END_OF_FILE* fires the transition *EJECTING_PCB* and inhibits component data readings of the both arms by means of inhibitor arcs. Only the data reading is inhibited, both of the arms are continuing their operation until their cycles are completed. After this, the completed PCB is ejected and, if there are more PCBs to assembly, (which is denoted by a token in the place *MORE_TO_DO*), the feeding of a new, unpopulated PCB is started. This transition consumes also the initial token from the place *OK_TO_ASSEMBLY*. When a new PCB is fed and fixed on the work table, the file pointer is reset and the token causing the data readings to disappear is consumed by the transition *RESETTING_FILE_POINTER*. Now, both robot arms are free again to continue their component assembly operations.

2) The left side of the lower part of the model in Fig 1 controls the pick and place operations of the arm_1. The initial marking is a token in the place *CYCLE*

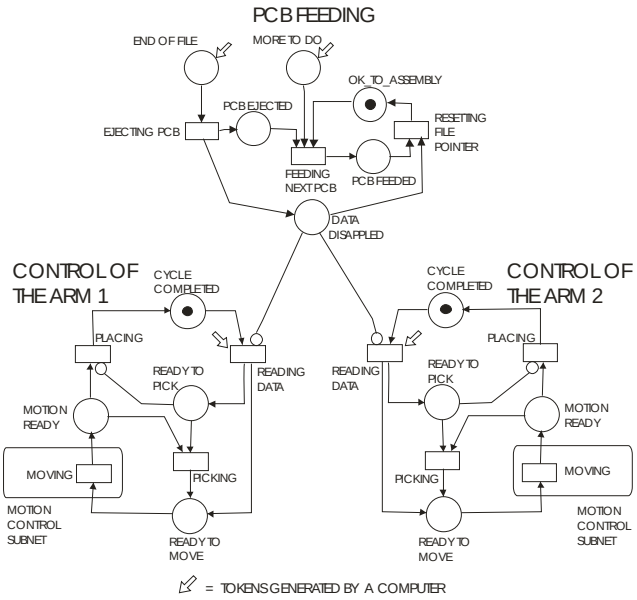


Figure 1 The virtual dual-handed robot to be modeled.

COMPLETED. This is also the place where the arm control waits after the EOF is reached and the assembly of the current PCB is completed. In that case the assembly controller disables the reading of data by means of an inhibitor arc. If the reading of data is not disabled and the pick-and-place cycle is completed, the next record of the data file is read.

The record contains the pick and place addresses for the next component placement. The transition *READING_DATA* then consumes the token from the place *CYCLE_COMPLETED* and creates tokens in the places *READY_TO_PICK* and *READY_TO_MOVE*. The token in the place *READY_TO_PICK* disables the transition *PLACING*. Therefore, after the arm movement has been completed, only the pick-up operation can be performed. One token in the place *READY_TO_MOVE* fires the transition *MOVING*.

Transition moving is a subnet and the details of the actual movement are defined and controlled in a subnet called "Motion control subnet", see Fig 1. While the arm performs two different kinds of movements, move-to-feeder and move-to-PCB, there is only one transition *MOVING* in the net. The motion is here always from the current position to the target position. Reaching the target position is notified by creating one token in the place *MOTION_READY*.

The alternation between the two kinds of movements is achieved by the use of exhibition arcs. If the transition *PLACING* is disabled by the inhibitor arc (as is the case at the beginning of a pickup-placement step), the next transition is *PICKING*. The transition *PICKING* consumes the tokens from the places *READY_TO_PICK* and *MOTION_READY*, and

creates one token in the place *READY_TO_MOVE*.

After the movement to the feeder is completed, there is again one token in the place *MOTION_READY* and the movement is from the feeder position to the placement position. This time, the transition *PLACING* is no longer disabled and *PICKING* is not enabled because there is no token in the place *READY_TO_PICK*. The next transition is therefore *PLACING*. After the component has been placed there is again one token in the initial place *CYCLE_COMPLETED*.

3) The right side of the lower part of the model in Fig 1 controls the pick and place operations of arm_2. The description of the model is analogous to that of arm_1. Note that both arms have their own motion control subnets.

Both arms operate independently; the only controls between them are the inhibitor arcs pointing to the transitions *READING DATA*. These two inhibitor arcs function only as on/off switches to enable and disable data readings, there is no other control for the arms at this main level. Further, there is not any mutual exclusion or other explicit collision detection or collision avoidance features in Fig. 1. It may seem that a robot controlled by the presented Petri net would collide and have a breakdown immediately, but this is not the case.

The explanation for this is that a hierarchical Petri net model has been used to model the robot. A more detailed description of the motion transitions, *MOVING*, is given in a subnet called "motion controller" in Fig. 2. The Petri net performs all movements in the x- and y-directions by taking small, user-defined steps towards the target position. The host net of the motion controller has only two places related to the motion: *READY TO MOVE* and *MOTION READY*. The host does not even know how the movement is actually performed.

The Petri net of the motion controller (Fig. 2) is almost symmetrical; the left side controls the x-movement and the right side the y-movement. Both directions are controlled concurrently and independently, so one direction may become *READY* while the other direction is still unfinished.

The collision manager operates as follows. Initially, after separating the x- and y-motions, there are tokens in the places *X-MOTION* and *Y-MOTION*. We follow here up only the x-movement because the y-motion is analogical. There are three possible situations: first, the target position may be reached and the motion is ready, otherwise (secondly), the robot hand has to move towards the target. The movement can be *HOMEWARD* (smaller reach-out) or *OUTWARD* (more reach-out). The third situation is, that the collision manager (described below) has detected a collision and takes partly over the motion control of both directions x and y. This "taking over" is performed by the means of placing tokens into the places *HOMEWARD*, *HALT_X* or *HALT_Y* according to the current collision situation. Other motions (not halted or forced to be *HOMEWARD*) continue under the control of the motion controller towards the target position. After each motion step, the token is delivered back to the place *X-MOTION* until the target position is reached.

The motion controller performs collision free movements. Collision freeness is achieved by using a subnet "collision manager". Both robot arms have their own copies of the motion controller, but only one common collision manager.

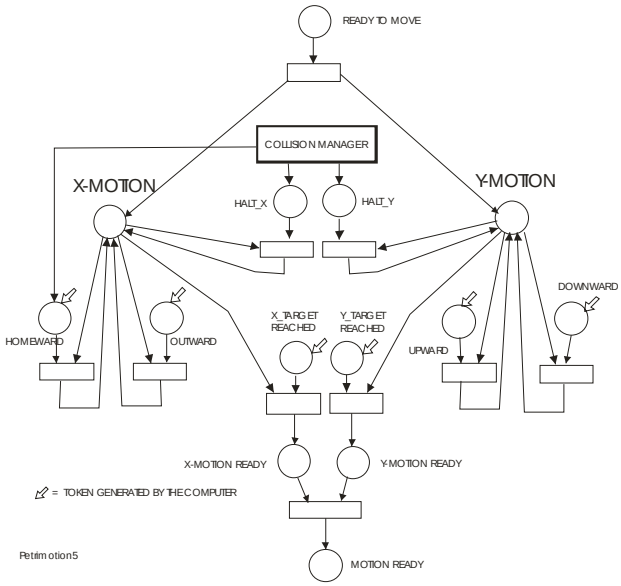


Figure .2 Motion controller.

3.2 Handling of the collision situations

Collision avoidance of concurrent, conflicting systems can be managed prior to the actual processing by issuing the proper scheduling and sequencing tasks for both arms. Optimal collision avoidance is, however difficult, indeed it belongs mathematically to the set of NP-complete problems [4, 5]. By “optimality” we mean in this case a plan with minimal operation time. Another, much easier way is to omit the collision avoidance calculations and instead of that to detect and manage collision situations on-line in real time.

In the case of an insertion machine for electronic components the prior planning for motions is impossible for several reasons. Some of these are: handling of erroneous components, some unexpected situation like run-out of some component type etc. We therefore consider in this subsection models for

collision management from the point of view of our virtual machine. Our interest is in the formulation of a Petri net model, which can be used for the timing analysis of a real target machine with similar operations.

In particular, we suppose that the robot includes some system for determining in real-time distances from each arm to the other or a common control system for both arms, which has real-time knowledge of the arm distances.

3.2.1 Assumptions

In order to simplify the simulator design we make three simplifications regarding the organization of the virtual robot.

1) Only the 2D case is concerned. In PCB assembly, most collision problems are in (x,y)-space. There are, however, some important exceptions, which should be kept in mind here. If the layout of the PCB is very tight, the order of insertions may be critical: a small component must sometimes be inserted prior to a large component in its vicinity. These types of constraints may cause problems for a two-handed robot, in particular if the precedence constraints deal with components placed with different arms. We suppose that a proper component-to-arm mapping and sequencing of the placements have solved these kinds of difficulties.

2) 2D linear/cartesian robots are used. Instead of scara or other more complex robot types, we suppose a robot with two linear arms. This excludes the possibility for hooking of the both arms together and largely simplifies the determining of the colliding parts and the time of collisions. Further, we can avoid a detected collision simply by stepping back, towards the home position, which is reducing the arms reach out. The home position is the minimum reach out position and it is supposed to be outside the maximum reach out of the other robot arm.

A linear robot arm here consists of two standard linear movement units connected together. The base arm is fixed to a robot body and cannot move. The moving arm is riding on the base carrier arm (y-direction) and it can move (reach out) orthogonal to the base arm (x-direction). A gripper or a nozzle to pick up components is fixed to one end (opposite to the driving unit end) of the linear unit.

3) The arms cover the components during the movement. All components to be picked and placed are supposed to be small enough to be inside of the arms borders. Thus, a component cannot collide against the component in the other arms gripper or against the other arm during arm movement.

3.2.2 Collision detection for the virtual assembly robot

The collision detection is easy for linear/cartesian robots, because the possible collision points are obvious. After a safe distance is defined, it is only necessary to continuously check the x- and y-directions (or radius for a round safety area) whether or not the physical borders of the arms violate the safety distance. This can be done by the test: **if** (*DX* > *SafetyDistance*) **or** (*DY* > *SafetyDistance*) **then** no collision **else** collision.

Here *DX* and *DY* stand for the distances between the arms in the x- and y-directions, as shown in Fig. 3. *SafetyDistance* is a user-defined distance between the borders of the arms. It has to be defined so that the movements of both arms can be slowed down and stopped before a physical collision damages the robot arms. The above collision detection decision can be performed step-by-step when advancing from a current position to the target position. The length of each step has not to be the same as that of the stepper motor but then one has to select the *SafetyDistance* so that it corresponds to the step size, velocity and deceleration capabilities of the robot arms.

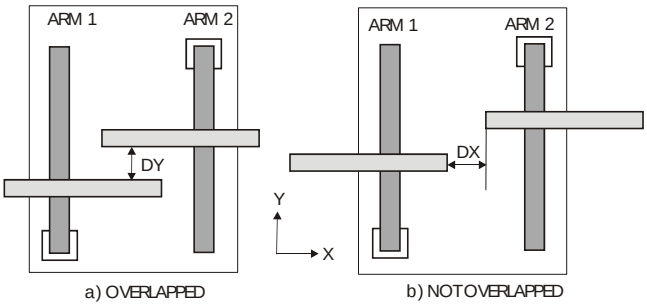


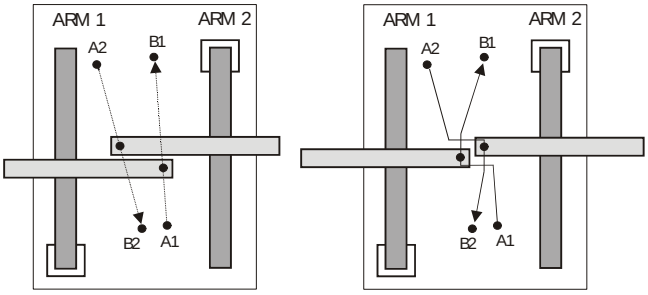
Figure 3 Possible collisions by a 2D cartesian robot.

3.2.3 Procedure to manage the collision situations

Procedure "blind courier" mimics the actions taken by a blind human delivering packets from one address to an other (a similar procedure is described in the artificial intelligence literature under the notion stimulus-response agent [3]). Because the courier cannot see, the person has to use hands or a stick to find by touching possible obstacles on the route. Before a step towards to the target address is taken, a check should made whether there is an obstacle (at the length of a hand or of the stick) on the route or not. If not then a step forward is taken. If an obstacle is found, side steps are made until the courier feels that there is no more an obstacle in the direction of the target. After this, stepping proceeds towards to the original destination. Because of the side steps the courier will be further from the original path, and a

new shortest path to the target position should be recalculated. In the dual-handed robot case, the only obstacle is the other arm (i.e. the other "Blind courier"). Both couriers behave the same; they take side steps towards home in case of an obstacle. The only difference comes from that the homes of the first and second couriers are on the left and right sides of the working area, respectively. Thus one courier side-steps to the left and the other to the right. The procedure is illustrated in Fig. 4, where Arm 1 (2) will be moved from A1 (A2) to B1 (B2).

While the procedure described above may work for some time, it is not sufficient for controlling the robot because it is not deadlock free. A deadlock happens when both couriers want to pick up or deliver at the same address at exactly same time. Both of them are then forced to make side steps but they cannot reach the target. Introducing priorities to the arms can solve this situation. The courier who is already closest to the target has priority to complete his task first. Now, after this addition the procedure may work some time longer but it still has a problem: should it happen that the both couriers are at exactly same distance from the target, then the deadlock is there again. This problem can be solved for example by saying that in such situation the courier_1 has the priority and the courier_2 has to wait until the courier_1 finishes his task.



a) Original tracks

b) Tracks due to collision avoidance procedure

Figure 4 Collision avoidance by sidestepping.

The collision management procedure described above is modeled as a Petri net for the dual-handed robot control in Fig. 5. The upper part of the figure, of inside the square, is the collision manager Petri net. The lower part of the figure, outside of the square, shows the places of the motion controllers of the both arms, in which the collision manager generates tokens.

The collision management starts its operation by testing: **if** (*DX* > *SafetyDistance*) **or** (*DY* > *SafetyDistance*) **then** no collision **else** collision.

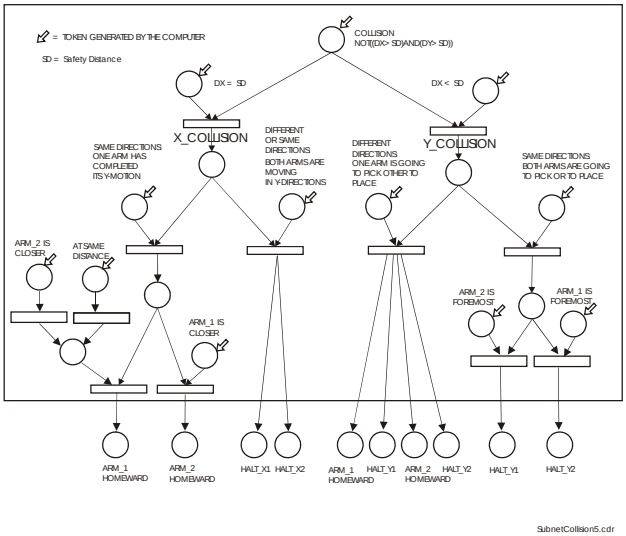


Figure 5 Collision manager subnet.

If there is a collision situation, a further test is made to decide between a x-collision and an y-collision. In a x-collision, a movement of the one arm in x-direction or the movements of both arms in opposite x-directions would cause a collision. The case of y-collision is analogous. If *DX* = *SafetyDistance*, (Fig. 3.b), there is a x-collision, otherwise if *DX* < *SafetyDistance*, (Fig. 3.a), there is an y-collision.

Let us consider the collision management in case of a x-collision; see the left side of Fig. 5. The operation of this net contains already plenty of functionality which is, however, relatively easy to interpret. There are two different situations. First, as long as both arms are moving in the y-direction, parallel in the same directions or in opposite direction, x-movements are restricted (no x-movement or no more reach out) until there is no more x-collision. This is achieved by sending the tokens, HALT_X1 and HALT_X2, into the corresponding places of the motion controllers of both arms. This is the right side arc under the place *X-COLLISION*. If both arms are moving in parallel in the same y-direction, it can happen that one arm completes its y-movement and stops while the other arm continues its y-movement and the x-collision situation is solved in this case.

But secondly, if both arms complete their y-movements in a x-collision situation, there might be a deadlock situation. This is the left side arc under the place X-*COLLISION*. If both arms have not completed their x-movement (because of the x-halts forced by the collision manager) and, if the left arm wants to move to the right and the right arm wants to move to the left, then one of the arms has to step homeward to allow the other arm to complete its x-movement. The rule here is that the arm closest to its target position has the priority and the other arm has to

take steps homewards. The deadlock, where both arms are exactly at the same distance from their targets, is solved by defining that the left arm has the priority. The homeward steppings are performed by sending tokens, *ARM_1 HOMEWARD* or *ARM_2 HOMEWARD*, into the corresponding place of the motion controller of the corresponding arm. The other arm continues at the same time its scheduled movement towards the target position.

Next, we follow the y-collision situations on the right side of Fig. 5. First we consider the left side arc under the place *Y-COLLISION*, see Fig. 4 for this case. The collision is solved by sending repeatedly the tokens ARM_1 HOMEWARD, HALT_Y1, ARM_2 HOMEWARD and HALT_Y2 to the corresponding places of the corresponding motion controllers until the collision situation is over. Second, consider the right side arc under the place *Y-COLLISION*. Here the foremost arm has completed its y-movement and it stops, and the other arm is still continuing its y-movement. To avoid the threatening collision, the collision manager send a token *HALT_Y1* or *HALT_Y2* into the corresponding place of the corresponding motion controller to halt and to force the other arm to wait. So, the foremost arm can complete its task without collision

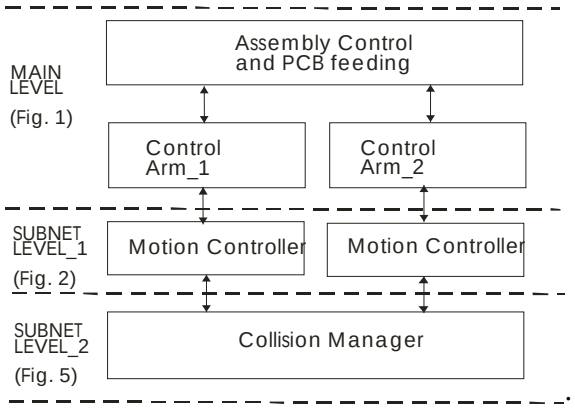


Figure 6 Overview of the complete Petri net model for robot control.

To sum up, the Petri net model of the dual-handed assembly robot consists of the main net, two instances of the subnet motion controller and the subnet collision manager serving both motion controllers. An overview of the complete Petri net is presented in Fig 6.

4. Animation of the Petri Net model

A software system was constructed to test, simulate and animate the function and the collision avoidance method of the robot. The system visualizes the operation of the virtual robot (Fig. 7 left) for a set of assembly instructions read from a file (list of components to be assembled with pick and place position addresses).

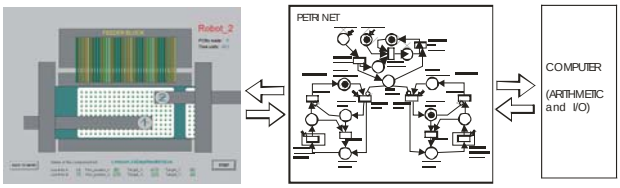


Figure 7 Virtual assembly system with a Petri net control.

To create a complete operating assembly system, we have congregated the animated virtual robot (on the left), the Petri net model (in the middle) and a computer together (on the right) as in Fig. 7. The computer is needed to simulate the Petri net and to perform the arithmetic arm position and distance calculations and file operations. The large hollow arrows present information flows between the component systems.

To realize the Petri net control system (parallel operations) on a standard computer (one processor and serial operations) it was necessary to define a serial order for the parallel operations. The serial processing of this Petri net control system a continuous loop. The processing begins by the PCB feeding and assembly instructions. After this the control of Arm_1 and then the control of Arm_2 follows. Finally, the control goes back to the beginning and the execution of the loop starts again. This serial operation does not harm the simulation and animation results in this kind of mechanical operations, where the physical actions are very slow compared to some microseconds the computer needs to make the calculations. The refreshing of the computer screen is not made until the control loop has been completed. Therefore, an observer sees the robot animation as if it were processed in parallel and all the motions were happening at the same time.

We fix the dimensions of the animation model and the speeds of the arm movements as follows: The length unit of the model is a pixel on a 1024 x 768 display. The shared work space (PCB and the picking zone of the feeder block having 38 feeders) is x = 420 and y = 280 pixels. The total movable area (shared area and private home area) is x = 500 and y = 280. Thickness of the robot arm is y = 60 pixels. The length of one step of the arm movement (in x- and y-directions) is chosen to be 10 pixels. The speed is then the number of steps in a time unit. The performance of the robotic assembly system was measured by counting the number of steps needed to complete the assembly task. Therefore, the actual

speed of the arms on the screen is not important for the performance calculation. The speed was chosen to suit observer's eyes in the animation. The *SafetyDistance* can be chosen freely as a given number of steps. The influence of the length of the *SafetyDistance* to assembly performance was studied in the simulations by varying this distance from 0 to 7 steps. The pixel values used here, can be converted to the corresponding real world values by using some scaling factor.

4.1 Simulated strategies

The simulator implements three different robots having three different behavioral strategies: one handed operation, two-handed in-turns operation mode and two- handed with real-time collision avoidance operation mode.

Robot 1: To set a basis for the working performance comparisons, the operation of one arm is switched off. So, the robot 1 works as an one-handed robot. The performance of the other robots are compared to this robot type.

Robot 2: The robot models the concurrent operation of the two hands having "in-turns" operation strategy. When one hand has finished its picking or placing task and *is away* from the shared space, the other hand gets the permission to enter the shared space. The hands use the "mutual exclusion" operation principle. There are no collision situations, but the performance of the machine is supposed to be lower due to the longer waiting times. The operation of this robot is very similar to that presented by Zhou [2].

Robot 3: Concurrent operation of the two hands having the collision and deadlock avoidance procedures was realized here. Petri nets for this robot are shown in Fig. 1, "motion controller" in Fig. 2 and "collision manager" in Fig. 5. This robot does not need any collision avoidance instructions or calculations made in advance, it detects and solves the collision situations autonomously in real-time. The influence of the length of the *SafetyDistance* to the assembly performance was studied by varying this distance from 0 to 7 arm movement steps.

To perform a simulation, also data sets (list of materials) are needed. We created three different (for 100, 200 and 300 components were to be selected randomly and placed on a PCB) data sets using a pseudo-random number generator and with different seeds. Each element of the set consists of three pseudo-random numbers: one for the feeder selection, second and third for the placing position (x and y) on a PCB. All three robots used these data sets as assembly orders/tasks. Robot 3 performed these tasks seven times using different *SafetyDistances* to verify how this distance influences the assembly performance.

4.2 Results of the Simulation and Performance Analysis

The results of the simulations are presented in Fig. 8. Beginning from the left, there are the performance bars of the one handed operation, which are scaled to value 1, to serve as a basis for the comparisons. The performance bars for the in-turns type of operation are next. Finally, there are the seven sets of performance bars for the operation with the real-time collision avoidance.

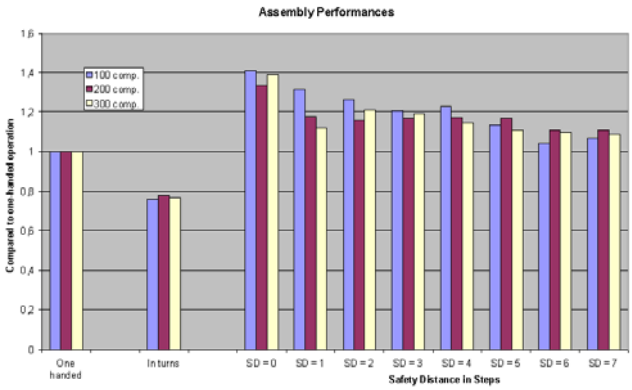


Fig. 8 Simulated performances for problems with 100, 200 and 300 components with safety distance (SD) ranging from 0 to 7 steps.

The results of the performance analysis were rather unexpected. The assumption before the analysis was that the performance in both two-handed cases (robot 2 and robot 3) would be approximately the same. Namely, robot 2 ("in-turns") had to wait until the shared workspace was safe to be accessed and robot 3 ("real-time collision avoidance") had to use longer paths to avoid collisions by taking side-steps. However, this assumption was wrong as we can see in Fig 8. The performance of robot 2 was lower (roughly 0.75 times) than that of the one armed robot. This means that the "in-turns"-operation of two arms is less advantageous than one arm operation. However, the performance of robot 3 was better than (roughly 1.2 times) that of the one armed robot. And robot 2 compared to robot 3, was roughly 1.2 / 0.75 = 1.6. Thus, in our test case the "real-time collision avoidance"-strategy was 60% more efficient than the "in-turns"-strategy.

The length of the safety distance influenced the assembly performance significantly. The best performance was achieved, having no safety distance at all (safety distance = 0). The virtual size of the robot arm is then the same as the real size of the arm.

As the safety distance was increased step by step, the performance decreased. The limit seems to be the performance of the one armed robot. The explanation for this is the strategy we have chosen for the dead-lock avoidance. As mentioned before, in a dead-lock situation, the arm being closest to the target position has the priority to continue and the other arm has to wait. Having a safety region around the arm means, that the virtual size of the arm is larger than the real size. Our animated simulation showed, that this results in a situation where one arm occupies the whole shared workspace and is always "closer" than the other, which has to wait aside at the home position. The two-handed robot operates then as one-handed robot.

5 Conclusions

A Petri net model for a concurrent procedure to avoid collisions of the two hands working concurrently on the same work area was developed in this work. The main idea was to not make the calculations for the collision avoidance prior to the operation. The need to make such kind of calculations off-line would reduce the flexibility of component assembly. Instead, a real-time collision detection and management procedure was suggested. The function of the procedure has been visually tested by the means of real-time animation. The simulation of the performance of the model shows unexpectedly that the performance of the "in-turns"-operation of two hands is less than that of one-handed robot. However, the suggested procedure presented in this work results in a gain of about 60% in performance compared to the commonly used "in-turn" type of operation. A control system using the suggested method can control dual-handed robots in a more flexible way and can achieve noticeable performance gains.

The simulation results presented in Section 4 can not be seen as a general truth. They are only valid for the here given geometrical dimensions, only one feeder block and pseudo random data. Other conditions like other geometry, two feeder blocks (one for each arm) sorted/optimized data etc. would produce totally different results.

References

[1] Sciomachen Anna and Grotzinger Stephen, *Petri Net-Based Emulation for Highly Concurrent Pick-and-Place Machine*, IEEE Transactions on Robotics and Automation, Vol. 6, No. 2, pages 242-247, April 1990.

[2] MengChu Zhou and Ming C. Leu, *Modeling and Performance Analysis of a Flexible PCB Assembly Station Using Petri Nets*, Transactions of the ASME, Vol. 113, pages 410-416, December 1991.

[3] Nils J. Nilsson, *Artificial Intelligence - A new Synthesis*, Stanford University, Morgan Kaufman Publishers, Inc., pages 21-35, 1998.

[4] Randal Wilson, Lydia Kavraki, Tomás Lozano-Pérez and Jean-Claude Latombe, *Two-Handed Assembly Sequencing*, International Journal of Robotics Research, Vol. 14(4), pages 335-350, 1995.

[5] Lydia Kavraki and Mihail Kolountzakis, *Partitioning a Planar Assembly Into Two Connected Parts Is NP-Complete*, Information Processing Letters, Vol. 55, pages 159-165, 1995.

[7] Petri Nets World, Petri nets homepage, University of Aarhus, Denmark URL.: http://www.daimi.au.dk/PetriNets/

[8] MengChu Zhou and Kurapati Venkatesh, *Modeling, Simulation and Control of flexible Manufacturing Systems a Petri Net Approach*, World Scientific, 1999.

[9] Wolfgang Reisig, *A Primer in Petri Net Design*, Springer Verlag, 1991.