1.0 Introduction

Autonomous systems have to guarantee safety while coping with execution in various environments particularly when robots interact with humans. These are the critical systems that need to be working without any failure. Failure to execute on the right time and place may lead to damage to humans or even death, or financial loss. To carry out robot work it needs to deploy a wide variety of software components such as planning, perception and safety critical components. Safety can be defined in absolute terms as “the absence of catastrophic consequences on the user(s) and the environment”[1]. Safety critical software involves diagnosis and fault detection and isolation, execution monitoring and isolation. A system is said to be autonomous when it has the ability to execute action autonomously also satisfying varying requirements demanded by the software. The varying requirements are difficult to track or predict due to the change in the environment feature, tasks and goals in real time. The changing requirements affect the functional components of the system and also safety critical components that help to handle the risk thereby protecting the system.
In the field of robotics deployment is considered as an ongoing activity in real time that expects redeployment of functional and safety critical software depending on the requirement posed by the system. Autonomous behaviour of the system expects the deployment of software components by themselves without any help from the external agent. By doing so human intervention is minimized and that justify the name autonomous.
Robotics deployment is already available in terms of architectural and platforms models[2,3] and configuration files[4,5]. It is found in [6] that the implementation of the deployment is done offline rather than run time changes. To solve the real time deployment of software components a reference archietcture[6] is taken that can deploy the software components in different platforms. The reference archietcture is realised in this paper for a collaborative robot application and validates the functioning of the system. Real time autonomous deployment of the safety critical software with varying requirements is realised in this paper.

2.0 Motivation

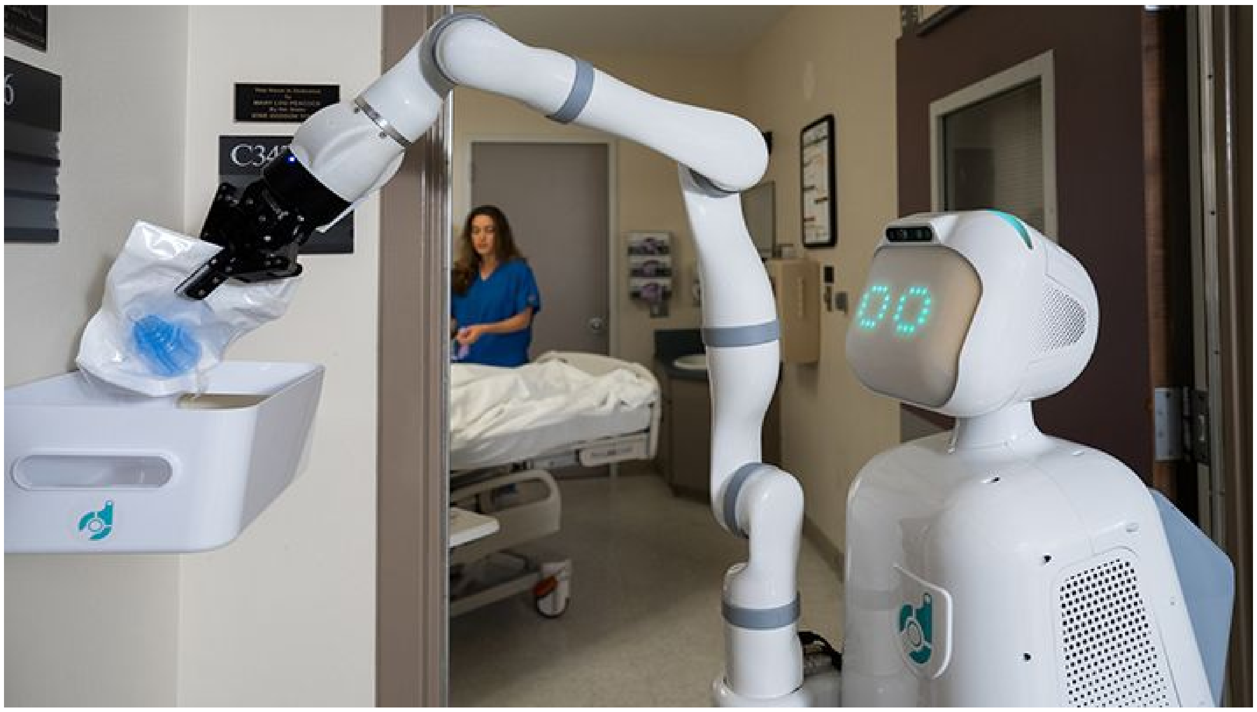

Fig 1.0

A collaborative robot helping humans to do their task is taken as an example in our case as shown in fig 1.0 to explain the deployment of safety critical softwares. Collaborative robots are a relatively new invention in the robotics community and nowadays they are widely used in industrial [7] , home and office environments. Traditionally collaborative robots existed only in the factory environment and are separated from direct contact with humans for safety reasons. In modern days collaborative robots are expected to work along with humans to carry out a wide variety of tasks from assisting with small tasks such as picking a fallen target material to helping to move huge objects. The close interaction with humans demands high safety standards over the life cycle of the robotics applications. ISO TS 15066[8] and ISO 12011 [9] standards help the safety critical engineers to perform the risk assessment of the collaborative robots while performing different tasks in different environments. Autonomous robot helper (fig 1) a collaborative robot is taken as the example to show the application area of deployment of safety critical software. The autonomous robot is capable of performing a wide variety of tasks such as preparing, transporting and handling different common things in a hospital environment. Risk assessment needs to be carried out on such robots to ensure safety. Risk can be in any form for example robots could damage a glass by dropping a glass while it tries to keep it on the table. So it is important to assess the risk, namely the safety monitoring system as explained in the below section.

2.1 Safety monitor for detecting glass slippage

Collaborative robots are equipped with different kinds of sensors to detect slippage to safely execute the manipulation task such as picking an object, placing and human handover. That is to detect whether the object is slipping from the robot's grasp. To detect the in hand slippage on the autonomous robot a architecture is proposed by Sanchez that has three different types of slip detectors based on tactile and force measurements and combination of these two sensors. The force slip detector detects a slip whenever there is a force exerted in the right direction. The tactile slip detector estimates the tangential force on the sensor caused by a sliding pressure. The combined slip detector combines the signal received by both tactile and force slip detectors in a rule such that the signal from force and tactile sensor is above a certain threshold value. These three slip detectors represent an active safety feature called as safety monitors that enables by checking the safety relevant information. Action that leads to danger or harm for example dropping of glass while a robot tries to place it on a table can be avoided by deploying safety critical software on the target platforms. Three slip detectors come into picture in different parts of the time and their performance is affected at the same time for example when the robot is in moving condition. Thus the evaluation of the autonomous robot suggests that all the action needs to be considered for improved safety performance. Hence it is necessary to adapt the safety monitor to the current robot’s actions at run time.

Framing deployment planning as a CSP

There are six requirement types that need to be considered to ensure deployment of the right safety monitor on the right platform. Attributes are namely quantity, capacity, minimum, maximum, attributes and selection. These requirements need to be framed as a constraint satisfaction problem in the minizinc environment. Threshold for different properties of the platform is initially declared and the values are passed onto the minizinc environment from the python environment. Three main thresholds include minimum force sensor, tactile sensor count and minimum memory fused together. Platform is defined as an array of platform numbers and their corresponding properties. Each row of the platform defines an individual platform along with their features. First column of the platform is the platform number, the second column corresponds to the force sensor status and can take a binary value to represent the status. Third column represents the number of active tactile sensors. Fourth column represents the memory availability belonging to a particular platform. Fifth column is the type of force sensor that is available in the platform and the final sixth column represents the latency speed availability in certain platforms. Platforms are assigned to every sensor. We make sure that the platform names are to be in the provided table. It is also made sure that values that need to be assigned to the platform names are available in the platform table by defining it as a constraint in the minizinc environment. In the technical constraint for force platform the force sensor value is greater than the threshold force sensor value passed from the python environment and speed in the force platform also needs to be greater than threshold speed. Type of force sensor used is also defined as a constraint. In similar fashion the tactile and combined system memory is defined by defining them as corresponding platform values greater than the threshold values. Once all the constraints are defined, a solve satisfy keyword is called to solve the constraint problem in the minizinc environment.

Conclusion

Safety monitoring systems for runtime is a critical component of the autonomous system. This article proposed and realised an approach to deploy the safety critical software on an virtual autonomous robot platform in real time with varying requirements posed by the system. This online deployment of safety monitoring strategy requires safety properties that need to be checked. This paper provides a solid approach to define the requirement types as a constraint and solve the constraint with the help of a constraint solver software.

Reference:

1. A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic ˇ concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, 1:11–33, 2004
2. Dhouib, S., Kchir, S., Stinckwich, S., Ziadi, T., Ziane, M.: RobotML, a domainspecific language to design, simulate and deploy robotic applications. In: Noda, I., Ando, N., Brugali, D., Kuffner, J.J. (eds.) SIMPAR 2012. LNCS (LNAI), vol. 7628, pp. 149–160. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34327-8 16
3. Nordmann, A., Hochgeschwender, N., Wigand, D., Wrede, S.: A survey on domain specific modeling and languages in robotics. J. Softw. Eng. Rob. 7(1), 75–99 (2016).
4. Bruyninckx, H., Soetens, P., Koninckx, B.: The real-time motion control core of the OROCOS project. In: Proceedings of the IEEE International Conference on Robotics and Automation (2003)

5. Quigley, M., et al.: ROS: an open-source robot operating system (2009)
6. Hochgeschwender, N., Biggs, G., Voos, H.: A reference architecture for deploying component-based robot software and comparison with existing tools. In: 2018 Second IEEE International Conference on Robotic Computing (IRC), pp. 121–128, Jan 2018.
7. Villani, V., Pini, F., Leali, F., Secchi, C.: Survey on human-robot collaboration in industrial settings: safety, intuitive interfaces and applications. Mechatronics 55, 248–266 (2018)
8. Robots and robotic devices - Collaborative robots. Technical specification, International Organization for Standardization, Geneva, CH (2016)
9. Safety of machinery - General principles for design - Risk assessment and risk reduction. Standard, International Organization for Standardization, Geneva, CH (2010)
10.