ORIGINAL ARTICLE

Requirements-Based Testing for Robotic Wheelchair Tracking Software System

Dayang N. A. Jawawi[1], Rooster Tumeng[1], M. Irsyad Kamil R.[1], Amirul Danish Misri[1], Rosbi Mamat[2], Shahliza Abdul Halim[1], Raja Zahilah[1], Mohamad Noor Hakim Mohamad[3]

[1] School of Computing, Faculty of Engineering, Universiti Teknologi Malaysia, Skudai 81310, Johor, Malaysia
[2] School of Electrical Engineering, Faculty of Engineering, Universiti Teknologi Malaysia, 81310 UTM Johor Bahru, Johor, Malaysia
[3] Hospital Sultanah Aminah, Jalan Persiaran Abu Bakar Sultan, 80100 Johor Bahru, Johor, Malaysia

ABSTRACT

Introduction: A robotic wheelchair system enables people with disabilities and elderly people who are unable to operate traditional wheelchairs to move around. Advances in computer and software technologies have contributed to the provision of robotic wheelchairs and its tracking system to suit the needs of the healthcare sector. Developing a quality software system requirements specification and performing testing on the requirements are critical as the users involve patients who suffer musculoskeletal disorders. **Methods:** This research paper presents a robotic wheelchair and its software system requirements; a testing environment; and a requirements-based testing strategy to ensure the software system comply with the requirements including the functional and technological constraints of the wheelchair system. **Results:** The testing strategy was implemented to calculate Average Percentage of Faults Detected (APFD) and the result quantify 92.19% APFD rates of fault detection with 92.19%. This result delivers an effectively improving test coverage. **Conclusion:** The proposed testing environment and the requirements-based testing strategy enable requirements testing to be conducted from the very beginning of the software development life cycle.

Keywords: Robotic wheelchair, Software requirements, Requirements-based testing, Simulation testing environment

Corresponding Author:
Dayang Norhayati Abang Jawawi, PhD
Email: dayang@utm.my
Tel: +607-5538768

INTRODUCTION

A robotic wheelchair (RWC) is a wheeled-motorized chair, incorporates hardware and software systems that include its own embedded computer and a software control system, that is designed for patients suffering from musculoskeletal disorders (1). There are many kinds of interfaces used to control robotic wheelchairs, as highlighted in Kamil et al. (2). Patients with special needs like those who are diagnosed with congenital disabilities or mobility impairment require a non-manual interface driver such as voice recognition, head gesture, or eye tracking. These requirements are essential towards the development of a RWC as it will increase the usability of the wheelchairs. Besides, the embedded computer in the RWC allows it to interact with other devices.

A camera connected to Wi-Fi may provide a 24-hour live streaming for the administrator's view, which allows them to monitor their patients. Furthermore, cameras can also pinpoint a patient's location based on the path trajectory made by the patients (3). This system requirement proves that RWC are able to contribute more not only to assist patients but also, monitor them.

Tracking is important in controlling wheelchairs since it will allow hospital personnel to monitor the current location of the wheelchair along with the current condition of patients. The retrieval of an object's exact location is effortless as Global Positioning System (GPS) technology is able to pin point to a most accurate position. However, in indoor positioning such as in hospitals, GPS accuracy is compromised as GPS discovery and positioning modules are obstructed by walls and ceiling, consequently, preventing the GPS from transmitting its location to satellites. In Zhang et al. (4) work's, it locates or tracks objects using a Wi-Fi positioning

technique and they proposed a selection method on access points to improve indoor Wi-Fi positioning accuracy. Meanwhile, Challa et al. (5) developed and implemented an indoor positioning system based on Bluetooth beacons, utilising the trilateration principle to aid in localization.

During the life cycle of RWC software and system development, errors may inevitably be introduced related to the system specification as well as attributed to human factor errors. Therefore, testing must be carried out at each stage to reduce error propagation. While numerous wheelchair systems have been developed, only a few are functional (6) discuss verification and validation activities of the system, especially the software testing. The requirements specification phase of RWC software development is critical because it can influence or constrain the software's design later. Performing testing on requirements for RWC is critical as the users involve patients who suffer musculoskeletal disorders.

Requirements-based testing refers to performing tests through deriving and executing test cases to ensure that requirements comply with the software that is developed (7). Requirements-based testing strategy allows the integration of testing throughout the early development life cycle. The strategy focuses on requirements specification's defect prevention, which leads to improving software quality. The purpose of this article is to propose a requirements-based testing strategy for verifying RWC software requirements in a testing simulation environment.

MATERIALS AND METHODS

The system and software requirements

An initial prototype of a robotic wheelchair was constructed to help our research at Universiti Teknologi Malaysia (UTM), with the primary goal of developing a minimal cost and an accessible system that supports various hardware and software experimentations (8). In this paper, we extend the requirements of the system to support Hospital Sultanah Aminah, Rehabilitation Department's services.

The RWC is visualized in Figure 1(a) as a block diagram consisting of a standard manual wheelchair and an add-on unit equipped with a single-chip microcontroller labelled the Embedded Processor. This version of our RWC supports the user interface and navigation using an Android device, and tracking and monitoring using a camera. Figure 1(b) shows an overview of a RWC solution proposed.

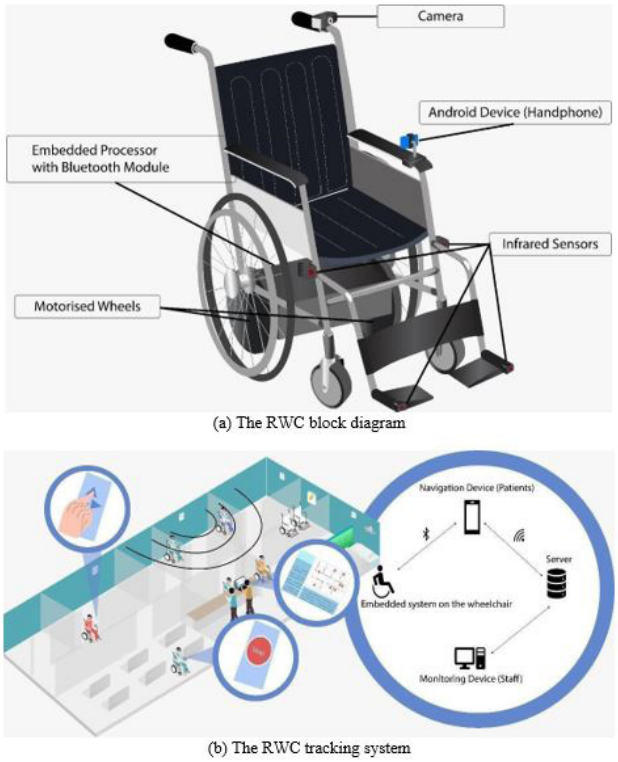


Figure 1 : The proposed RWC System. (a) The RWC block diagram, **(b)** The RWC tracking system.

Here, we proposed a solution using Bluetooth and Wi-Fi technologies to fulfil RWC tracking and navigation requirements. The system's tracking design consists of four major components: an embedded processor/computer for the wheelchair, an Android device, a server and a monitoring device as shown in Figure 1(a).

The Android device will be used to communicate with the RWC embedded computer via a Bluetooth connection. The communication will involve signals for navigation and retrieving information of the wheelchair. In addition, data received by the Android device will be recorded and uploaded via the internet to the server. Furthermore, the recorded data will be retrieved by hospital personnel to be displayed and for monitoring purposes. Through harnessing a Bluetooth Low Energy Beacon (BLE Beacon) technology, the position of the wheelchair can also be monitored by the hospital personnel. In addition, a live streaming feed provided by a camera attached to the wheelchair will also help in providing a rich visual information of the wheelchair's conditions or operations. The feed also helps the personnel in identifying the condition of patients. The RWC will automatically send a

notification to the tracking system whenever the RWC encounters any errors.

In terms of artificial intelligence (AI) implementation, Google Cloud Vision API will be used to recognise the text from the frames of the streaming video (camera). The result and data of the recognition will be used by the system to process to obtain the coordinate and determine the current location of the wheelchair. The result of the data processed, the camera system will save the data to the database. The tracking system will receive the stream of the latest coordinate and location of the wheelchair to be displayed together with the frame of the video stream that has been sent through the web-socket by the camera system.

The software requirement documentation

In real world scenarios, software requirements are often captured and structured in natural language that aid in understanding but these could result in inconsistencies. High-level requirements are presented in the form of natural language that are prone to issues of incompleteness, inconsistencies, and difficulty of interpretation. In our works, there are four high-level requirements.

I. This system must be able to control wheelchair navigation.
II. The system must be able to avoid obstacles found in the path.
III. The system must be able to monitor the position of the wheelchair.
IV. The system should be able to display the current live video monitoring

Towards designing the RWC implementation, high-level requirements must be refined further with more details into low-level requirements. An instance of refined low-level requirement for the first requirement that could be transformed into formal specification is specified in Table I. The pre-condition for the requirement is that the system is already connected to the RWC embedded software. From this low-level requirement, the control behaviour of the RWC system could be designed. In ensuring that the RWC tracking system requirements are met, low-level requirements could be used to specify test cases, which could then be used to test whether the design is in compliance with the requirements.

The testing simulation environment

This section describes how a simulation environment works to represent the RWC requirements, described in the actual RWC as discussed in the previous section. A major difference in this simulation environment is that the camera is not attached to the RWC due to the small size of the developed model. Instead, a camera will be used as a computer vision processing to obtain the position of the RWC, which acts as a BLE Beacon in the real implementation.

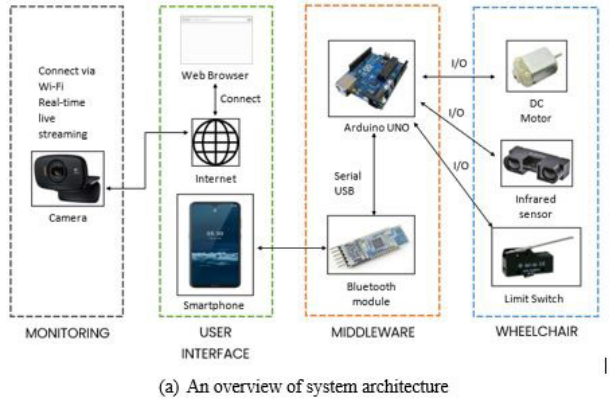
Table I : Low-level requirements for ID1 Control Navigation requirement.

ID	Description
1.1	The patient selects navigation mode. If the patient selects 'Manual Navigation', refer to alternative flow (a. Manual Navigation). If the patient selects 'Semi Auto Navigation', refer to alternative flow (b. Semi Auto Navigation). **Alternative Flow (a.) Manual Navigation :** (i.) The patient selects navigation mode'. (ii.) The patient controls the wheelchair. **Alternative Flow (b.) Semi Auto Navigation :** (i.) The patient locates the current position. (ii.) The patient selects destination. (iii.) The patient presses start.
1.3	The system will retrieve path movement.
1.4	The system saves the generated path movement.

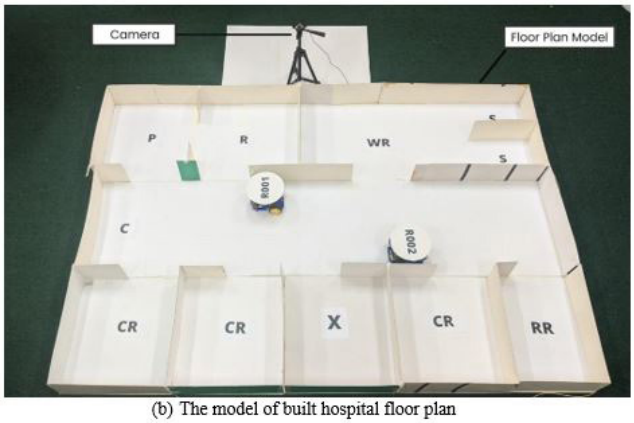
In this simulation prototype model, a mobile robot, equipped with two motors represents the implementation of a RWC. The system architecture of the simulation prototype is described in Figure 2 together with a hospital plan model where the simulation runs. The four layers of the system architecture are depicted in Figure 2(a), namely monitoring, interface, middleware, and wheelchair. The user interface is a mobile application that assists users by displaying context-sensitive accessibility resources based on their displacements. Furthermore, users control their wheelchair via mobile application. The application is connected to the wheelchair via Bluetooth. Signals emitted and received are saved in a text log, which will assist in testing later. For the wheelchair, there are three main components comprising DC motors, infrared sensors, and limit switches. DC motors and wheels move the robot while infrared sensors and limit switches are both used to detect obstacles near the wheelchair. The function of the infrared sensor is to sense any obstacles within its surrounding. Meanwhile, the limit switch assists in collision detection with obstacles. The sensors are triggered upon object detection in front of the robot. The robot will consequently stop and maintain an idle state regardless of the user's commands.

In the RWC prototype, an Arduino microcontroller is used as the embedded computer. It is designed to control the directions and speeds of the RWC by simply receiving input commands via the Bluetooth module from the user interface. The prototype is also designed to first check obstacles. If there are no obstacles, the command will be accepted and the RWC will run according to the command.

As for the mobile controller, the purpose of the mobile application is to allow users to control the RWC prototype from their own mobile phones via Bluetooth. The system requires a camera and an internet connection to provide live streaming and broadcasting purposes. The camera is equipped with a camera software to recognize robots and their respective plate numbers. The mobile controller controls the robot movements while the camera feed tracks the robot's movements. The camera is mounted on a tripod stand to track all of the robot's movements as shown in Figure 2(b). A program coded in Python tracks the robot's movements by marking them with a frame. Besides tracking the robot, the program is also able to identify the location of multiple robots along with their respective robot IDs. This function aims to benefit users for the real implementation as it will allow medical personnel to track all wheelchairs in the hospital.



(a) An overview of system architecture



(b) The model of built hospital floor plan

Figure 2 : System architecture and prototype environment model. (a) An overview of system architecture, **(b)** The model of built hospital floor plan.

RESULTS

In this section, the proposed work revolves around deriving test cases and testing the requirements of the implemented requirements in the RWC simulation environment to improve software quality via improving fault detection.

The proposed testing strategy is a new technique of prioritizing test cases based on improved string distance through considering requirements priority weight. A complete illustration of the testing strategy is shown in Figure 3. It is aim to assign weights to requirement priorities such as weights assigned by developers and managers. As the wheelchair is still undergoing an active development phase at UTM lab, customer priority is omitted in this study. The improved string distance is based on multiplication of Smith Waterman (9) string distance and average requirement priority weight.

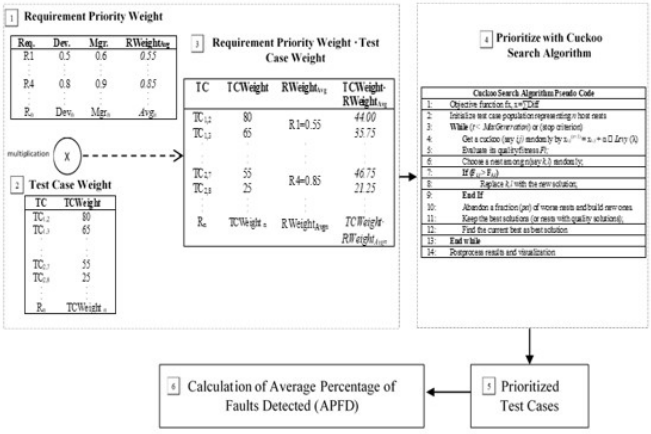


Figure 3: Requirements-based testing strategy.

Figure 3 : Requirements-based testing strategy.

Based on Figure 3, the testing strategy comprises six steps. The first step concerns averaging requirement priority weights from developer and manager. The second step involves calculating test cases weights via Smith Waterman string distance. Eq. (1) expresses matrix filling scoring that is contained in Smith Waterman string distance. Eq. (2) will result in unnormalized Smith Waterman distance value, unnormalisedSW.

$$Score(i,j) = Max\{Score_{(i-1,j)} - P,\ Score_{(i,j-1)} - P,\ Score_{(i-1,j-1)} + cost \qquad (1)$$

where,

$Score_{(i-1,j)}$	is the scoring matrix to be filled;
i,j	are matrix's indices;
$cost$	comprises three preset values; 0 if $T_1.length() < T_1[i]\ T_1[i] < 0,\ T_2.length() < 1$ if character at index else -2.0
P	is a penalty for a gap, set at 0.5.

$$SW(T_1,T_2) = \left(\frac{unnormalizedSW}{maxScore} \right) x\ 100 \quad (2)$$

where,

$SW(T_1,T_2)$ is Smith Waterman distance computed between test cases T_1 and T_2,

unnormalizedSW is un-normalized value of Smith Waterman distance,

maxScore is maximum edit score between T_1 and T_2,

The implementation of Smith Waterman string distance utilizes an open source library developed by Chapman (10), known as Simmetrics in Java platform.

The third step involves simple multiplication of requirement priority and Smith Waterman distance which yields the proposed improved Smith Waterman distance. The fourth step, meanwhile, involves prioritizing the test cases through utilizing Cuckoo Search Algorithm (CSA). The CSA algorithm is to mimic parasitic behavior of certain Cuckoo bird species that seek to lay their own eggs on host nests.

Cuckoo Search Algorithm Pseudo Code

```
1:  Objective function fx, x=∑Diff(xi,j, ...,Xin,xjn)~(xpoor_tc_i, poor_tc_j)
2:  Initialize test case population representing n host nests xi,j (i = 1, 2, ...,n; j = 1, 2, ...n);
3:  While (t < MaxGeneration) or (stop criterion)
4:      Get a cuckoo (say i,j) randomly by xi,j^(t+1) = xi,j + α ⊕ Levy (λ)
5:      Evaluate its quality/fitness Fi;
6:      Choose a nest among n(say k,l) randomly;
7:      If (Fi,j > Fk,l)
8:          Replace k,l with the new solution;
9:      End If
10:     Abandon a fraction (pa) of worse nests and build new ones.
11:     Keep the best solutions (or nests with quality solutions);
12:     Find the current best as best solution
13: End while
14: Postprocess results and visualization
```

Figure 4 : Cuckoo search algorithm.

Figure 4 outlines CSA pseudo code based on Dhareula and Ganpati work (11).

Referring to Figure 4, the objective function Ledru et al. (2012) (12) of this CSA, as expressed in Eq. (3), is to keep the sum of largest distance difference, Diff, of current best test case and requirement priority, against current poor test case and requirement priority, i.e. test case with the lowest current weight as the current best solution using Eq. (3).

$$ObjectiveFunction(T_{(i,j)}) = \sum_{i=1)} Diff\,[(T_{i,j}) - (T_{(poortc\_i,poortcj)})] \quad (3)$$

where $i=1,\ldots,n$

$j=1,\ldots,n$ and $i \neq j$

The fifth step will be to visualize the prioritized test cases. The test cases are ranked according to the CSA search algorithm's best solution output. Finally, the Average Percentage of Faults Detected (APFD) of the prioritized test cases is calculated. The equation of APFD is shown in Eq. (4).

$$APFD = 1 - \frac{TF_1+TF_2+TF_3+....+TF_m}{nm} + \frac{1}{2n} \quad (4)$$

where,

TF_1 is the first index of fault revealing test case in *i*,

n is the number of test cases,

m is the number of mutants or seeded faults.

Tracking system prototype was used for evaluation in this study. For the system, a manual test case generation was designed based on use cases. Results show 32 test cases were generated for three use cases of the prototype. All 32 test cases are extracted from the tracking system requirement. An example on a test case extracted can be referred to Table II. For validation, two versions of the prototype were created. A unique fault was seeded into each of the prototype versions. In version 1, there are nine faults revealing test cases. Meanwhile, in version 2, there are also coincidentally nine faults revealing test cases.

DISCUSSION

In overall, 32 test cases were created from the RWC Tracking prototype. The initial test suite, T_0 holds the test cases, that is not sequenced in any specific orderings. In each version, one fault that is unique was seeded. The T_0 is then executed using CSA. In total, 32 nests were produced upon subjecting T_0 to CSA. Each nest comprises 19 CSA-ordered test cases. The 19 test cases ordering based on CSA are then re-ordered based on requirement priorities and the ordering result are shown in Table II along with APFD.

Underlined test cases in Table II signify first test cases that identify seeded faults. Since there are two seeded faults, there are two underlined test cases. APFD of the three nests are shown in Table II. Based on results glanced in Table III, out of 32 test cases – only 19 test cases are selected as best solutions while remaining 13 test cases have been eliminated.

APFD in Table II indicate that prioritized test cases with improved Smith Waterman distance and requirement priorities utilizing CSA algorithm performed with evident efficiency. Out of the three nests, Nest 3 exhibits the highest APFD score. This hints that such a nest yields the best ordering of test cases. From the results, it is safe to assume that Nest 3 is chosen as the most ideal set of CSA-prioritized test cases, T_{CSA} = {T18, T9, T12, T6, T25, T5, T21, T17, T6, T8, T31, T7, T2, T19, T14, T29, T20, T10, T4}.

Table II : Example of Extracted Dataset from Requirement

SWS R1: Control Navigation:: Manage Control						
Requirement: User select navigation mode to manoeuvre a wheelchair						
Test Case ID#	Bluetooth	Battery	IR Sensor	Limit Switch	Right Motor	Left Motor
TC001	Connected	Available	On	On	On	On
TC002	Connected	Unavailable	Off	Off	Off	Off
TC003	Disconnected	Available	On	On	On	On
TC004	Disconnected	Unavailable	Off	Off	Off	Off

Table III : Results of RWC Tracking System

TF*i*	1	2	3	4	5	6	7	8	9	10	APFD
Nest 1 =	[T18,	T19,	T12,	T23,	T17,	T21,	T4,	T1,	T7,	T31]	0.8438
Nest 2 =	[T16,	T5,	T13	T1,	T21	T4,	T22,	T14,	T20,	T7]	0.7813
Nest 3 =	[T18,	T9	T12	T6	T25	T5	T21	T17	T16	T8]	0.9219

The seeded fault 1, F_1, can be detected by nine fault revealing test cases, TF_1 = {T24, T25, T26, T27, T28, T29, T30, T31, T32}. The first TF_1 that reveals fault in $CSA_{ordering}$ is at index 5, which is test case T25. Similarly, seeded fault 2, F_2, can be detected by nine fault revealing test cases, TF_2 = {T15, T16, T17, T18, T19, T20, T21, T22, T23}. The first TF_2 in $CSA_{ordering}$ that reveals fault is at index 1, which is test case T18. From these TF_1 and TF_2 values, APFD is computed, yielding a value of 92.19%. The 92.19% reflects the rate of fault detection per percentage of test suite execution T_{CSA} prioritized test cases. This testing strategy provides quantitative tests within the proposed six steps, ensuring that testing is being adequately performed during the simulation testing.

CONCLUSION

This paper provides a robotic wheelchair system requirements, describes a testing simulation environment to implement the functionality of the requirements and proposes a requirements-based testing strategy to quantify the quality of faults detected based on software requirements based on the environment. Finally, the testing strategy was implemented to calculate APFD and quantify the rates of fault detection based on the requirements and prioritized test cases. It delivers maximum coverage with a minimum number of test cases while also effectively improving test coverage. The testing simulation environment and requirements-based testing strategy proposed in this paper is a desirable way to discover software problems as early on in the requirements engineering process. In this sense, a systematic early requirements-based testing is a critical step towards establishing a quality software system for robotic wheelchairs.

ACKNOWLEDGEMENT

The authors wish to acknowledge Universiti Teknologi Malaysia for UTM-TDR Grant Vot No. 06G23 and Ministry of Higher Education (MOHE), Grant Vot No. 5F117, which have made this research possible. The authors would also like thank members of Software Engineering Research Group for the continuous support and inputs provided for this research.

REFERENCES

1. Voznenko, Timofei I., Eugene V. Chepin, and Gleb A. Urvanov. (2018). The Control System based on Extended BCI for a Robotic Wheelchair. Procedia Computer Science, 123: 522-527.
2. Kamil, R. M., Jawawi, D. N. A., Mamat, R., and Jamal, N. N. (2020). Indoor Smart Wheelchair: Systematic Mapping. IOP Conference Series: Materials Science and Engineering, 864(1).
3. Ciuccarelli, L., Freddi, A., Iarlori, S., Longhi, S., Monteriш, A., Ortenzi, D., and Proietti Pagnotta, D. (2019). Architecture for Cooperative Interacting Robotic Systems Towards Assisted Living: A

Preliminary Study (A. Leone, A. Caroppo, G. Rescio, G. Diraco, and P. Siciliano (eds.); Vol. 544, pp. 471–486). Springer International Publishing.
4. Zhang, W., Yu, K., Wang, W., and Li, X. (2020). A Self-Adaptive AP Selection Algorithm Based on Multiobjective Optimization for Indoor WiFi Positioning. IEEE Internet of Things Journal, 8(3), 1406–1416.
5. Challa, N. S. R., Kesari, P., Ammana, S. R., Katukojwala, S., and Achanta, D. S. (2019). Design and implementation of bluetooth-beacon based indoor positioning system. 2019 5th IEEE International WIE Conference on Electrical and Computer Engineering, WIECON-ECE 2019 - Proceedings, Ld 33, 2019–2022.
6. Boucher, P, Atrash, A, Kelouwani, S. 2013. Design and validation of an intelligent wheelchair towards a clinically functional outcome. J Neuroeng Rehabil 10(1): 58.
7. Sommerville, I. (2011) Software Engineering. 9th ed. Boston: Pearson.
8. Jawawi D. N. A., Sabil S., Mamat R., Mohd Zaki M. Z., Talab M. A. S., Mohamad R., Hamdan N. M. & Kamal K. 2011. A Robotic Wheelchair Component-Based Software Development, Mobile Robots – Control Architectures, Bio-Interfacing, Navigation, Multi Robot Motion Planning and Operator Training, Book 2, Intech Open Access Publisher. ISBN 978-953-307-842-7, pp. 102-126.
9. Smith, T. F., and Waterman, M. S. (1981). Identification of common molecular subsequences. Journal of molecular biology, 147(1): 195-197.
10. Chapman, S. (2005) SimMetrics-open source similarity measure library. Retrieved Feb 01, 2021, from http://sourceforge.net/projects/simmetrics/.
11. Dhareula, A. and Ganpati, P. (2019).Cuckoo Search Algorithm for Test Case Prioritization in Regression Testing. International Journal of Recent Technology and Engineering, 8(3): 6004-6009.
12. Ledru, Y., Petrenko, A., Boroday, S., and Mandran, N. (2012). Prioritizing test cases with string distances. Autom Softw Eng, 19: 69-95.