# Modified Bug Algorithm with Proximity Sensors to Reduce Human-Cobot Collisions

Anran Li
*School of Engineering and Computer Science*
Washington State University, Vancouver, USA
anran.li@wsu.edu

Hakan Gurocak
*School of Engineering and Computer Science*
Washington State University, Vancouver, USA
hgurocak@wsu.edu

2023 9th International Conference on Automation, Robotics and Applications (ICARA) | 978-1-6654-8921-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICARA56516.2023.10126056

***Abstract*—This paper presents an adaptation of the mobile robot bug algorithm to control a cobot to reduce collisions with a human worker in the same work cell. Collaborative robots (cobots) are popular in industry because they enable close collaboration between a human and a robot. Although the cobot can stop when a collision happens, it is not a pleasant or safe working environment for the human to be hit by the cobot possibly multiple times a day. The proposed approach uses inexpensive proximity sensors on the cobot and a simple algorithm. An adjustable yaw angle is introduced to the algorithm to further reduce collisions. Results from pick-and-place experiments with UR 10 cobot show significant reduction of collisions.**

***Keywords—cobot, robot, collision avoidance, bug algorithm***

## I. INTRODUCTION

Collaborative robots (cobots) are widely used in industry to improve manufacturing efficiency. Human worker and cobot can work together to achieve complex tasks. Unlike traditional robots, if a collision happens between the cobot and the worker, the cobot can stop. But frequent collisions in a typical workday are undesirable since they lead to downtime and potential injuries. Therefore, it is desirable to reduce or avoid cobot collisions with the human worker.

In this paper, we present a real-time collision avoidance algorithm to prevent the cobot from colliding with the worker in the work cell. Multiple inexpensive proximity sensors were used to detect environment information and a novel low computing complexity bug algorithm was developed to reach a target object while significantly reducing or eliminating collisions with the human worker.

Literature contains various collision avoidance ideas. A common collision avoidance algorithm introduces an artificial potential field (APF) in a cobot environment. The target point generates an attractive force and obstacles generate repulsive forces which are applied to the velocity of the cobot. However, the APF method may drop into local minima when artificial resultant forces become zero. In [1-3], the method is improved with various approaches. [1] used two Kinect depth cameras to construct a 3D depth model to estimate the cobot location and obstacles. The APF resultant force is transferred into joint velocities and uses the Jacobian matrix and its inverse to establish a differential velocity function of joint motion [2]. The method was used for collision problems in a dual-arm robot system. A collision method was implemented by combining rapid random tree (RRT) algorithm with APF to try to eliminate the local minimum in a global map [3].

Another method for collision avoidance is the RRT algorithm. The RRT is a global path planning algorithm, which traverses all possible nodes randomly between a start point and target point then returns the shortest and collision-free path as a final path. Goal probability is used as bias and cost function for each random traverse, a node with a higher probability and lesser cost is selected as the next node [4]. The method reduces traverse time, and computational complexity, which is successfully applied to a dual-arm robot system. A slice-based heuristic fast marching tree was used to achieve a global collision-free path planning [5], it is combined with a cost function [6] to enhance the exploration efficiency. A variable step and target probability were used to offset RRT collision-free path planning method, an offset toward the target point promises its planning time is 70.05% less than the original RRT method [7]. In real applications, however, how to acquire global environmental information for path planning exploration is a critical challenge.

Bug algorithm is a collision avoidance algorithm used in mobile robots. Unlike the RRT, the bug algorithm is a local path planning method for tracking the boundary of an obstacle. [8-12] introduce improved bug algorithms as collision-free path planning solutions in a dynamic environment. The bug algorithm was combined with the RRT in [9]. An emergency level determines which algorithm is of top priority. IR sensors and ultrasonic sensors were used to achieve collision-free path exploration [10].

In addition, data-driven collision avoidance methods are widely applied. Different kinds of machine learning methods, combined with datasets collected from depth cameras and proximity sensors in robot arm systems were used, and collision cases were reduced by their training AI models [13-17]. Though the data-driven method is widely applied in recent collision avoidance research, its performance highly relies on the quality of the dataset.

Studies that use vision systems [1-7] can perform well but they demand high computational power to process images in real-time. Also, they plan the path based on a global environment where obstacle locations are known. Whereas

applications such as [13-17] focus on data-driven methods to learn the collision avoidance path. They highly rely on data quality, data content, and high computation resources.

In Section II, the new collision avoidance algorithm is presented. Section III presents the experiments and results with a UR-10 cobot by Universal Robots. Conclusions follow in Section IV.

## II. Methodology

### A. Experimental Setup

As shown in Fig. 1, the experimental work cell consists of a UR10 cobot on Table II, a target object on Table I, data acquisition (DAQ) card by National Instruments, multiple HC-SRO4 ultrasonic proximity sensors (Fig.1 right) and a PC.

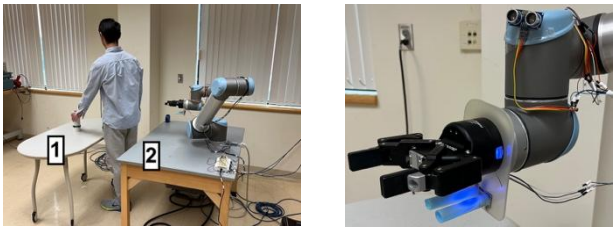


Fig. 1. Experimental workspace (left) and the proximity sensors under the fingers and on the wrist (right).

The front sensor below the fingers triggers the bug algorithm. There are three other sensors on the wrist pointing to the sides to prevent collision with the human worker or any other object in the environment.

In this application scenario, normally the worker processes a part (for example, by inserting a screw into it) then the cobot picks it up from Table I and delivers it to Table II. However, sometimes the worker comes in the way of the cobot as it tries to pick up the part from Table I. In this case, the cobot may collide with the worker and stop. Or, using the front ultrasonic sensors on its wrist, it can be stopped to wait for the worker to finish his processing and clear the path. But this causes downtime. Instead, the new approach can maneuver the cobot around the human to reach the part at the target position on Table I.

The PC acts as a server and implements the control logic (Fig. 2). It also monitors sensors using the DAQ and sends target end-effector positions for the UR10 to move. The PC server and the UR10 communicate through network socket API. An OR gate circuit was used to connect the echo signals of multiple proximity sensors to the same counter pin of the DAQ.

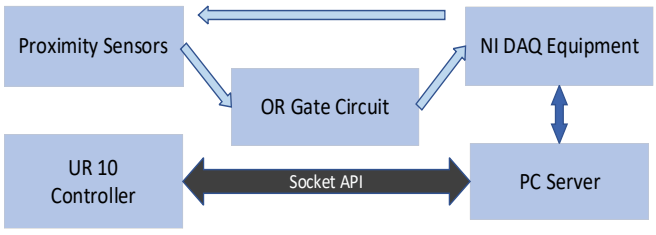


Fig. 2. Workspace hardware.

The UR10 starts at home position shown in Fig 1. Each pick-up and delivery is regarded as one experiment. The worker starts at the left end of Table I. Then, he moves right 2 inches in each new experiment toward the right end of the table. This leads to 29 different worker locations (experiments). If the cobot collides with the worker, the algorithm will stop searching for a solution and reset for the next experiment. This means the current position for the worker within that experiment produces a failed attempt to pick up the part.

The PC server implements multiple threads. Moreover, a dynamic sensor data stack was built to achieve real-time data updates. Fig. 3 and Fig. 4 show flowcharts for the cobot controller and the PC.

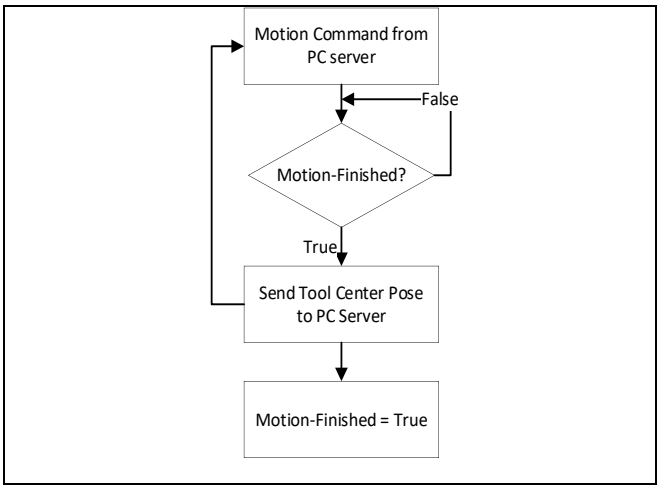


Fig. 3. Cobot controller program flowchart.

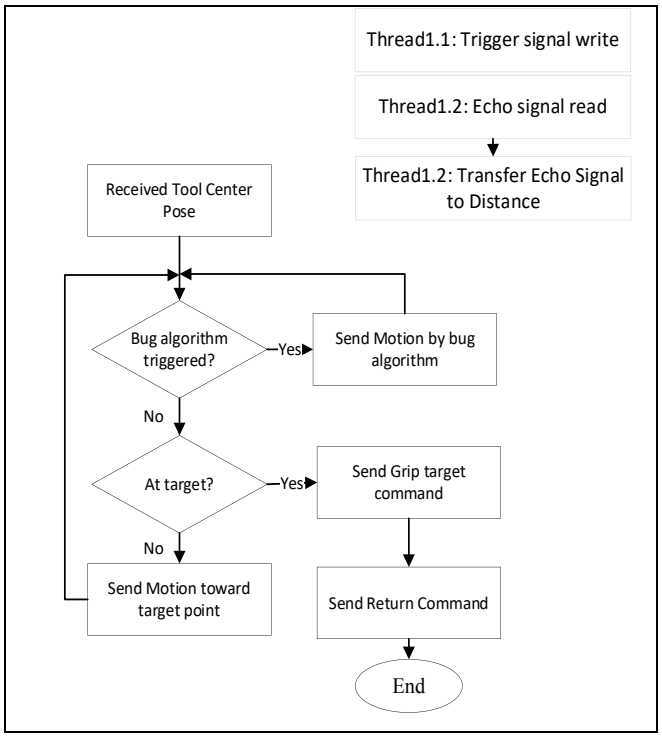


Fig. 4. PC server program flowchart.

The cobot controller has pose maneuvering, receiving, and sending functions. The cobot runs a motion command received from the PC server, once the motion is finished, current tool center pose is sent to the PC server and a motion-finish flag is set to true and waiting for the next command from the PC server. In Fig. 4, two threads are created in the main process to trigger proximity sensors and encode echo signals for distance values. The main program sends motion commands according to the bug algorithm operation, current tool center pose, and the sensor data. A command to move towards the target object continues to be sent to the cobot controller until the bug algorithm is triggered. When the tool center is at the target object, a gripper-close command is sent, and a return command ensures the cobot moves back to the start point following the collision-free trajectory.

### B. *Modified Bug Algorithm*

The bug algorithm is triggered if the worker is detected using the front proximity sensor. According to the bug algorithm simulation results in [18], the goal of the algorithm is to move the end-effector parallel to the worker's back to get around him to reach the targeted part

**Algorithm**

*front proxy* is triggered (high)
Cobot stops
Calculate $\theta_1$
**While** *front proxy* is high
  Move backwards away from the detected object (eq. 1)
  **If** *front proxy* is low
    Move to the right (eq. 2)
    Calculate $\theta_2$
    **If** $\theta_2 > \theta_1$
      Rapid move back to trigger position
      Move left (eq. 2)
      **If** distance traveled > expected travel distance
        Rapid move to trigger position
        Move right (eq. 2)
      **end**
      **If** distance traveled > full shoulder width
        No solution found,
        Stop movement, alert worker
      **end**
      **If** iterations >= iteration limit
        No solution found,
        Stop movement, alert worker
      **end**
    **end**
  **end**
**end**
**If** *proxy* is low
  *departure point* = end-effector position
  Calculate *yaw angle (or use fixed yaw angles)*
  End-effector frame orientation = yaw *angle*
**End**

To try to move towards the part, the algorithm first calculates the angle $\theta_1$ between the target position and the detected position as shown in Fig. 5. Initially, motion to the right is chosen as a direction and the position of the end-effector (E-E) is incremented. If the new angle, $\theta_2$ (Fig. 5) increases from $\theta_1$, motion to the left is chosen as a direction instead.

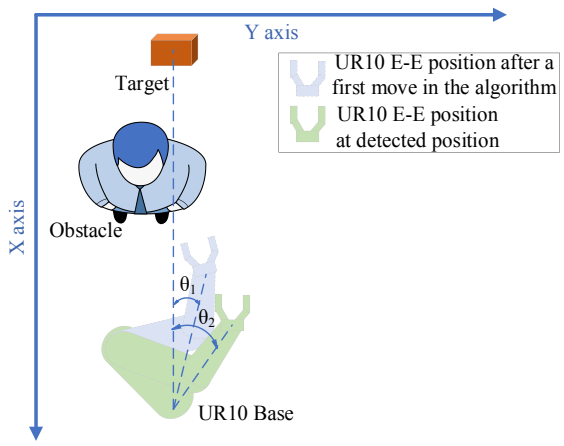


Fig. 5. Angles $\theta_1$, $\theta_2$ calculated within the bug algorithm to determine motion direction.

The condition to change motion direction is the expected travel distance, which is defined as half of the worker's shoulder width plus the proximity sensor detection radius and a tolerance of 2 inches.

The UR10 uses inverse kinematics to move. The bug algorithm features two different movements, the $\Delta z$, and the $\Delta X$ & $\Delta Y$ movement to maneuver the end-effector around an obstacle. The $\Delta z$ movement is with respect to the coordinate frame {8} of the sensor attached to the end-effector (eq. 1). Its purpose is to back up the end-effector to keep a safe distance away from an object once it is detected. The next movement of the end-effector, the $\Delta X$ & $\Delta Y$, is with respect to the base frame {0} of the UR10 (eq. 2). It moves the end-effector around the object to get a clear sight of the target position.

$$ {}^{8}P1 = \begin{bmatrix} 0 \\ 0 \\ -\Delta z \end{bmatrix} \quad (1) $$

$$ {}^{0}P2 = \begin{bmatrix} \pm\Delta X \\ \pm\Delta Y \\ 0 \end{bmatrix} \quad (2) $$

## III. Experiments and Results

### A. *Position Experiments*

The experiments start with the worker at the left edge (position 1) of table 1 with his shoulders parallel to the table. During the cobot motion to pick up the part, the worker remains at the same location. The experiments are repeated for 29 positions across the table until the worker is at the right edge of Table I.

As the worker moves across his table increasingly blocking the target position, the number of iterations jumps to 100 or close to it followed by a sharp decrease as the target is no longer blocked as seen in Fig. 6 (left). In Fig 6 (right), a dead zone is created when the worker is at the center of his table and the maximum iteration count is reached.

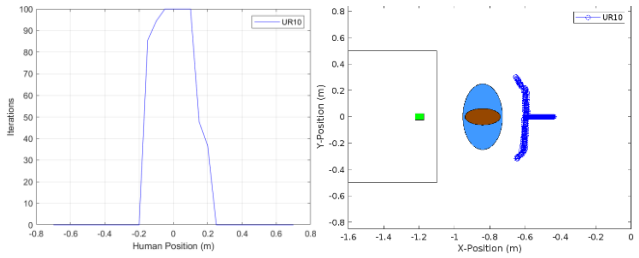


Fig. 6. The iterations through the experiments and an X-Y plot of the end-effector when the iteration limit was reached. (Left) shows the number of iterations in the bug algorithm. (Right) top view shows a dead zone that develops when the iteration limit is reached and the cobot stops.

Successful pickups occur when the worker is partially blocking or not blocking the part at all. The outcomes from the bug algorithm in 29 positions experiments can be seen in Table 1 compared to the same scenario performed with no use of the bug algorithm (baseline). There is no change in successful part pickups when using the algorithm but there is a large decrease in the number of collisions compared to the baseline experiments.

The algorithm stops in three conditions: (1) no solution found, (2) iteration limit reached, (3) collision is about to occur with any objects (side sensors). The iteration count was set to 100. If any proximity sensors detect an object within 2 inches of the arm, it will stop the cobot from collision. In Table 1, in the baseline case the cobot collides with the human 12 times and the side sensors trigger 4 times out of the 29 runs. With the bug algorithm, the cobot collides with the human only 3 times out of the same 29 runs. The side sensors trigger 9 times and in 4 attempts iteration limit was reached stopping the cobot.

TABLE I. BUG ALGORITHM POSITION EXPERIMENT RESULTS

| | ***Successes*** | ***Collisions*** | ***Side Sensor Detections*** | ***Iteration Limit Reached*** | ***No-Solution Counts*** |
|---|---|---|---|---|---|
| **Baseline** | 13 / 29 | 12 | 4 | N/A | N/A |
| **Bug Algorithm** | 13 / 29 | 3 | 9 | 4 | 0 |

In Fig. 7, the bug algorithm maneuvered the cobot around the right side of the worker when the worker stands at locations to the left of the target table (figure on the left). In the subsequent experiments where the worker got closer to the right edge of the table, the algorithm decided to move the UR10 around the left side of the worker (figure on the right), These feasible paths are explored based on the local real-time proximity information.

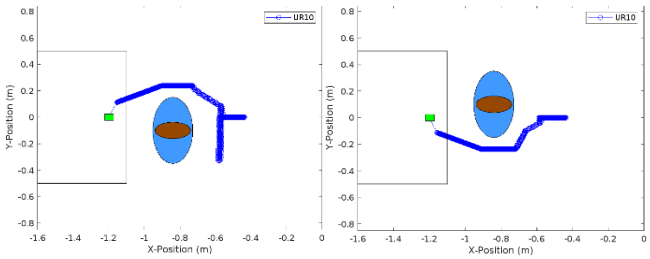


Fig. 7. Top view of the work cell with cobot end-effector motion path using the algorithm. (left) Cobot passes from the right side. (right) Cobot decides to move around the worker from the left side as the target is occluded more.

Fig. 8 shows a close-up of the end-effector positions for an experiment as the UR10 is moving around the worker. The first $\Delta X, \Delta Y$ movements by the algorithm can be seen in the red circle. The worker is detected at point 1 and $\theta_1$ is calculated. The end-effector is then moved backwards to position 2 and 3 (cyan circle in Fig. 8) until the proximity sensor is not triggered. A new exploration starts at position 4 and follows the trajectory to point 5. In point 5, an obstacle is no longer detected by the head proximity sensor, it is saved as the departure point and the cobot is moving forward to close the target position.

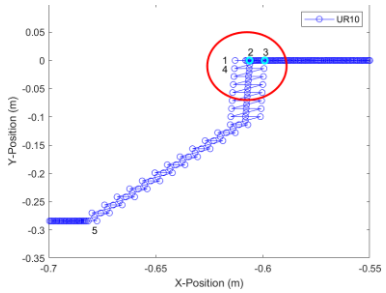


Fig. 8. Top view of UR10 movement around the worker showing details of the motion by the bug algorithm.

## *B. Human Worker Orientation Experiments*

In the previous experiments, the worker's shoulders were parallel to the table and his position across the table was changed to assess the bug algorithm. In this set of experiments, the effect of worker's orientation was explored. The worker stands at one of the 29 positions across Table I. Then, in each experimental run his angle with the table is changed by 15° increments at that position. Ultimately, the worker goes from facing table 1 to facing perpendicular to the Table I.

In Table II, the number of successful pickups increases as the angle increases from 13 to 17/29. However, the number of collisions increases when a worker's shoulders are at an angle larger than $30^0$ with the table. The ultrasonic sensor's detection angle is ±15°. Sensor detection error is introduced once the worker's angle is increased.

TABLE II. RESULTS FOR HUMAN WORKER ORIENTATION EXPERIMENTS

| **Worker's Orientation** | **Successes** | **Collisions** | **Side Sensor Detections** | **Iteration Limit Reached** | **No-Solution Counts** |
|---|---|---|---|---|---|
| **0°** | 13 / 29 | 3 | 9 | 4 | 0 |
| **15°** | 14 / 29 | 3 | 9 | 3 | 0 |
| **30°** | 15 / 29 | 5 | 4 | 5 | 0 |
| **45°** | 15 / 29 | 5 | 7 | 2 | 0 |
| **60°** | 15 / 29 | 4 | 8 | 2 | 0 |
| **75°** | 17 / 29 | 4 | 8 | 0 | 0 |
| **90°** | 18 / 29 | 2 | 8 | 1 | 0 |

## *C. Wrist Yaw Angle Experiments*

Up to this point, the cobot approached the target with its fingers perpendicular to the worker's table. In some cases, this causes a collision with the worker when the fingers clear the worker but the arm touches him. Hence, introducing a yaw angle allows the wrist of the cobot to rotate towards the target object as the cobot maneuvers around the worker almost like warping around the worker. The yaw angle was calculated in the

X-Y plane while in motion using the target part position and the departure point as shown in Fig. 9. The yaw angle idea was tested using (1) a calculated "departure" angle, and (2) using a set of fixed angles from 10° to 50° with 10° increments for $\theta$. A yaw angle of 50° puts the UR10 close to a singularity point of the arm as it was close to being fully extended.

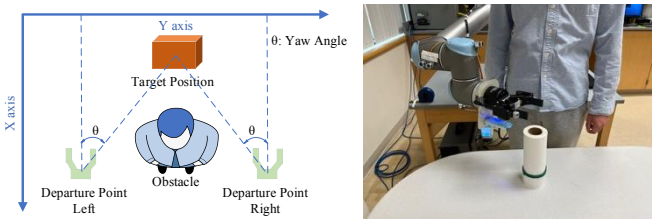


Fig. 9. (Left) Yaw angle calculation. The angle is measured using the departure point and the target part position. The departure point is the position of the end-effector when the algorithm successfully moves the cobot around an obstacle and the front proximity sensor is not detecting an object. (Right) Cobot successfully reaching around the worker with the new yaw angle.

In Table I, the bug algorithm kept the cobot's fingers perpendicular to the shoulders of the worker and achieved 13/29 pickups with 3/29 collisions. Table III shows that as the yaw angle increases, the number of successful pickups increases and the number of collisions goes down from 3/29 to 2/29. Overall, if a yaw angle of 30-40$^0$ is used, the modified bug algorithm with yaw angle can increase successful pickups from the baseline of 13/29 to 18/29 while significantly reducing the collisions with the human from the baseline of 12/29 to 2/29. For example, in Fig. 10 with no yaw angle, the UR10 stopped because of a side proximity sensor detection (left). But with the yaw angle, the UR10 could now pick up the part (right).

TABLE III. EXPERIMENTAL RESULTS FOR BUG ALGORITHM WITH YAW ANGLE

| Yaw Angle | Successes | Collisions | Side Sensor Detections | Iteration Limit Reached | No-Solution Counts |
|---|---|---|---|---|---|
| **0°** | 13 / 29 | 3 | 9 | 4 | 0 |
| **Departure** | 17 / 29 | 3 | 7 | 1 | 1 |
| **10°** | 16 / 29 | 2 | 9 | 1 | 1 |
| **20°** | 17 / 29 | 3 | 7 | 1 | 1 |
| **30°** | 18 / 29 | 2 | 6 | 2 | 1 |
| **40°** | 18 / 29 | 2 | 7 | 1 | 1 |
| **50°** | 20 / 29 | 2 | 5 | 1 | 1 |

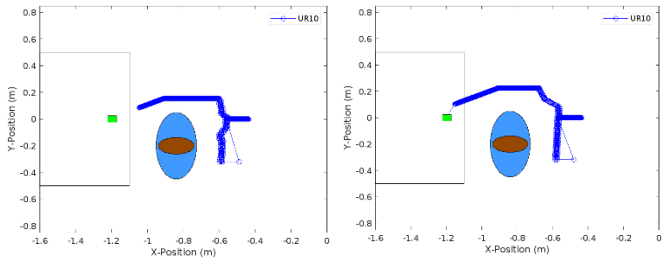


Fig. 10. Effect of the wrist yaw angle. (Left) Failed trajectory without yaw angle (object is not reached). (Right) Successful trajectory with yaw angle (object reached, thin line from the hand to the object in the figure for the final cobot move).


## ACKNOWLEDGMENT

Authors would like to thank the Amazon Robotics and AI laboratory for the UR10 used in this research.


## IV. CONCLUSIONS

In this research, we presented an adaptation of the mobile robot bug algorithm to control a cobot to reduce collisions with a human worker in the same work cell. The novelty of our approach is that using a simple algorithm and inexpensive sensors, the cobot can significantly reduce or eliminate collisions with the human worker in many instances in an unknown environment.

Another feature of the new algorithm is the wrist yaw angle, which creates a more feasible space, reduces the number of collisions, eliminates collisions in some cases, and increases the number of part pick-ups. Experiments showed that the wrist yaw angle should be set at least to the calculated departure angle in real-time. This is a lower limit for the yaw angle. The upper limit was bound at 50° due to the kinematics and work cell layout in this research. The new approach is a promising improvement to the cobot's capabilities in physical experiments as the number of collisions were decreased in all experiments compared to the experiments with an "out-of-the-box" cobot with no algorithm.

## REFERENCES


[1] Wang Z., Xu, H., Lü, N., Tao, W., Chen, G., Chi, W. and Sun, L., "Dynamic Obstacle Avoidance for Application of Human-Robot Cooperative Dispensing Medicines," in *Journal of Shanghai Jiaotong University (Science),* 27(1), pp.24-35, 2022.

[2] Zhang, H., Zhu, Y., Liu, X., and Xu, X. "Analysis of obstacle avoidance strategy for dual-arm robot based on speed field with improved artificial potential field algorithm," in *Electronics*, 10(15), 1850, 2021.

[3] Cheng, C., Lv, X., Zhang, J. and Zhang, M., "Robot Arm Path Planning Based on Improved RRT Algorithm." In *2021 3rd International Symposium on Robotics & Intelligent Manufacturing Technology (ISRIMT)*, pp. 243-247. IEEE, 2021.

[4] Shi W., Wang, K., Zhao, C. and Tian, M., "Obstacle Avoidance Path Planning for the Dual-Arm Robot Based on an Improved RRT Algorithm." *Applied Sciences* 12.8: 4087, 2022.

[5] Yu Y., and Y. Zhang. "Collision avoidance and path planning for industrial manipulator using slice-based heuristic fast marching tree." in *Robotics and Computer-Integrated Manufacturing* 75: 102289. 2022.

[6] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, 1985, pp. 500-505.

[7] W. Yang, W. Haiying, and Z. Zhisheng. "Obstacle Avoidance Path Planning of Manipulator Based on Improved RRT Algorithm." In *2021 International Conference on Computer, Control and Robotics (ICCCR)*, pp. 104-109. IEEE, 2021.

[8] K.N. McGuire, G.C.H.E. de Croon, K. Tuyls, "A comparative study of bug algorithms for robot navigation," in *Robotics and Autonomous Systems*, vol. 121, Nov. 2019.

[9] Y. Min, K. Wu and K. Li, "Online Trajectory Planning Strategy for UAV in Dynamic Threats Based on Bug-Rapidly-exploring Random Tree Algorithm*," 2021 China Automation Congress (CAC),* 2021, pp. 4814-4820, doi: 10.1109/CAC53003.2021.9728317.

[10] E. Ahmed AA, M. IAM Osman, N. AA Elhag, and M. A. Manzoul. "Sensor-Based Obstacle Avoidance for Autonomous Mobile Robots: Experimental Study." In *2021 7th International Conference on Automation, Robotics and Applications (ICARA)*, pp. 26-31. IEEE, 2021.

[11] F. Khan, A. Alakberi, S. Almaamari and A. R. Beig, "Navigation algorithm for autonomous mobile robots in indoor environments," in

*Advances in Science and Engineering Technology International Conferences (ASET)*, pp. 1-6, 2018.

[12] W. Kamil, T. Tarczewski, and Ł. Niewiara. "Local Path Planning for Autonomous Mobile Robot Based on APF-BUG Algorithm with Ground Quality Indicator. " In *Advanced, Contemporary Control, 979–90. Cham: Springer International Publishing*, 2020. https://doi.org/10.1007/978-3-030-50936-1_82.

[13] D. Apan, and M. Lin. "Dynamically Avoiding Amorphous Obstacles with Topological Manifold Learning and Deep Autoencoding." *arXiv preprint arXiv:*2203.13282, 2022.

[14] H. Jie, W. Ge, H. Cheng, C. Xi, J. Zhu, F. Zhang, and W. Shang. "Real-time Obstacle Avoidance in Robotic Manipulation Using Imitation Learning." In *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV),* pp. 976-981. IEEE, 2020.

[15] Abdel-Nasser Sharkawy and Nikos Aspragathos, "Human-Robot Collision Detection Based on Neural Networks," *International Journal of Mechanical Engineering and Robotics Research*, Vol. 7, No. 2, pp. 150-157, March 2018. DOI: 10.18178/ijmerr.7.2.150-157.

[16] K. Yeom, "Deep Reinforcement Learning Based Autonomous Driving with Collision Free for Mobile Robots," *International Journal of Mechanical Engineering and Robotics Research*, Vol. 11, No. 5, pp. 338-344, May 2022. DOI: 10.18178/ijmerr.11.5.338-344. .

[17] A. Ali, M. Ranjbar, and J. Park. "Computer vision-based path planning for robot arms in three-dimensional workspaces using Q-learning and neural networks." *Sensors* 22, no. 5: 1697, 2022.

[18] Cichosz, C. and Gurocak, H., "Collision Avoidance in Human-Cobot Work Cell Using Proximity Sensors and Modified Bug Algorithm," *The 9th International Conference on Control, Mechatronics and Automation (ICCMA), Luxembourg*, November 11-14, 2021.