Noname manuscript No.
(will be inserted by the editor)

Adding degrees of freedom to automated quantum Braitenberg vehicles

Nimish Mishra · Rayala Sarath Chandra · Bikash K. Behera[@] · Prasanta K. Panigrahi

Received: date / Accepted: date

Abstract Ever since practical quantum computation is possible, there have been attempts to apply the power of quantum mechanics to robotics and develop quantum equivalent of classical robots called quantum robots. Quantum robots, based on quantum logic, have the ability to exhibit truly non-deterministic behaviour based on quantum phenomena (superposition and entanglement), making them superior to their classical counterparts. As the degrees of freedom of such robots increase, the number of possible input-output combinations increase, making it difficult to derive circuits from arbitrary logic in a structured way. We attempt to provide a structured process for such derivations. To achieve this, we draw parallelism between classical Boolean logic and quantum propositional logic of eigenstates. We find that it is possible to treat quantum propositional logic of a quantum robot as classical Boolean logic, and that we can create equivalent quantum circuits to realize such logic. We introduce independent ideas: non-deterministic K-maps and n-qubit AND/OR operations with their efficient implementations, which might be of independent interest to the reader. We present algorithms used to realise

Nimish Mishra
First author, Department of Computer Science and Engineering, Indian Institute of Information Technology, Kalyani, 741235, West Bengal, India.
E-mail: nimish_bt18@iiitkalyani.ac.in

Rayala Sarath Chandra
First author, Department of Computer Science and Engineering, Indian Institute of Information Technology, Kalyani, 741235, West Bengal, India.
E-mail: rayala@iiitkalyani.ac.in

Bikash K. Behera[@]
Bikash's Quantum (OPC) Pvt. Ltd., Balindi, Mohanpur, 741246, Nadia, West Bengal, India
E-mail: bikash@bikashsquantum.com[@]Corresponding Author

Prasanta K. Panigrahi
Department of Physical Sciences, Indian Institute of Science Education and Research Kolkata, Mohanpur 741246, West Bengal, India
E-mail: pprasanta@iiserkol.ac.in

the ideas, demonstrate adding further degrees of freedom, and discuss adding non-deterministic behaviour. Our findings are a step towards our continued study on generalization of quantum logic for quantum robots.

Keywords IBM Quantum Experience, Quantum Robots, Automated Quantum Braitenberg vehicles, Finite Automata, Moore Machines, Degrees of freedom

1 Introduction

The idea of a theoretical framework for a quantum computer was first suggested by Paul Benioff in 1980 [1] amidst increasing inquiry into the physical nature of *classical* information first introduced by Rolf Landauer [2]. Feynman pointed to the intrinsic ability of a quantum computer (*if it could be constructed*) to handle matrix operations better than its classical counterpart [3], while Manin wrote on similar lines in his book *Computable and Uncomputable* [4]. Quantum Turing machines were then conceptualized by Deutsch [5].

The idea of a quantum robot was first introduced by Benioff [6] as a mobile quantum system coupled with a quantum computer and an ancilla system, that lets the robot take deterministic/non-deterministic action based on its measurement of the environment. A quantum robot is able to take advantage of purely quantum phenomena - superposition, entanglement, violation of Bell's inequality [7], and non-deterministic nature of quantum computation and quantum algorithms. Thereafter, several inquiries into this field have been made. Dong *et. al.* [8,9] worked on learning algorithms for quantum robots. Zeno *et. al.* formalized the use of *eigenlogic* for introducing special behavioral possibilities to the robot [10], and Zizzi [11] demonstrated a quantum metalanguage to control robots.

In this paper, we extend our previous work on non-automated quantum robot having three degrees of freedom in movement [12] and on automation of Braitenberg vehicles using finite automata Moore machines [13]. We demonstrate increases in degrees of freedom of movement in a quantum Braitenberg vehicles.

The paper is organized as follows. In Section 2, we present an introduction to classical non-automated/automated Braitenberg vehicles and discuss previous works, including that of the authors, done on the subject. In Section 3, we establish a theoretical reasoning on whether any arbitrary input-output relationship function may be realized for a quantum robot. In Section 4, we introduce ideas regarding non-deterministic K-maps and n-qubit quantum operations, demonstrate building a quantum robot using three degrees of freedom, and discuss the flexibility of a quantum robot in allowing specific intrinsic behaviour to be added to it.

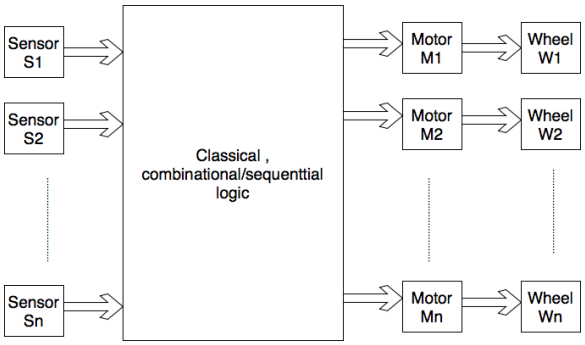

Fig. 1 Schematic of a classical Braitenberg vehicle. The sensors feed the inputs to the combinational/sequential logic. The motors receive the outputs and drive the wheels [13].

2 Automated classical Braitenberg vehicles

Braitenberg vehicles were first introduced by Braitenberg [14] as combinations of sensors and wheels attached to motors, all guided by electronic circuitry that activates certain motors based on sensor inputs (the sensors sense light as input). Different behaviours of the vehicles are manifested by different Boolean functions (sensor input- motor output relationships) implemented in the circuitry.

The above implementation in Fig. 1 has certain disadvantages: the need to continuously provide light signals as inputs, generating light signals of frequencies different from those already in the atmosphere, and preventing errors such as unintentional signals to a different sensor [13]. A plausible solution is to automate the vehicle using memory. We refer the reader to procedure of automating detailed in [13] and abstracted in Fig. 2.

We refer to Mishra *et. al.* [13] for details on minimized Boolean function derivation of the circuitry. The Braitenberg vehicle abstracted in Fig. 2 can be implemented as a quantum vehicle by involving quantum circuitry to compute the sensor input-motor output functions, as demonstrated in Mahanti *et. al.* [12] and Raghuvanshi *et. al.* [15]. Quantum Braitenberg vehicles have distinct advantages over their classical counterparts: it is possible to introduce desired intrinsic non-deterministic behaviour to the robot based on superposition and entanglement, and introduce external control over the operation of that intrinsic behaviour (demonstrated in [13]). Throughout the paper, we use $|0\rangle$ and $|1\rangle$ as indication of absence of light and presence of light respectively, in context of sensor inputs and as indication of no movement and forward movement respectively, in context of motors.

3 Degrees of Freedom

The quantum version of propositional logic was first introduced by Birkhoff and Von Neumann [16]. In this version, to each property E is attached a quan-

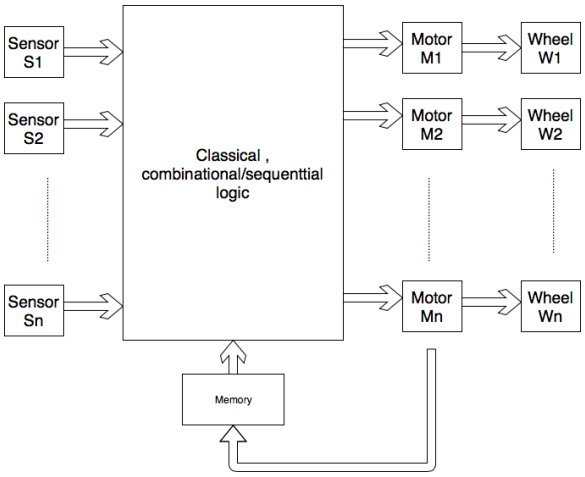

Fig. 2 Moore machine implementation of an automated Braitenberg vehicle. It can be observed here that how memory is a factor in the logic deciding the output states based on the inputs from sensors [13].

tity R such that measurement of R distinguishes the presence or absence of E. Therefore, any closed linear subspace corresponds to an elementary proposition (a true/false proposition). Co-measurability implies propositions can be measured by simultaneous measurements. Any two propositions, p and q are co-measurable *iff* there exists mutually orthogonal propositions a, b, c such that $p = aVb$ and $q = aVc$, where xVy denotes all vectors that are linear combination of x and y. The propositions we use in this paper correspond to the quantum states $|0\rangle$ and $|1\rangle$, denoting the absence and presence of light respectively for sensors and no movement and movement for motors. These propositions are co-measurable since they are orthogonal to each other and thus physically distinguishable in measurement.

This co-measurable nature makes it possible to treat quantum logic in quantum robots like classical logic and apply classical logic manipulations on the same (for instance, distributive law is applicable in this case due to no violation of uncertainty principle of measurement of two observables, and this is due to co-measurable nature of the observables in question). We further refer that any classical boolean function can be subjected to minimization techniques like Quine-Mccluskey method or the Karnaugh map method. Since quantum logic of quantum robots behaves as classical Boolean logic, the same minimization principles are applicable here. Classical Boolean logic may be sufficiently represented using basic gates - AND, OR, and NOT gates. We establish their equivalents in a quantum computer in Figures 3, 4, and 5. NOT operation is trivial.

We conclude, henceforth, that, theoretically, desired number of degrees of freedom can be added to a quantum robot, since it possible to construct circuits

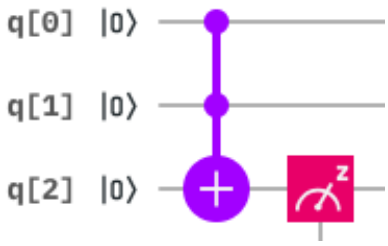

Fig. 3 Circuit for AND operation. The third qubit is the output qubit, resulting in eigenstate $|1\rangle$ when both the inputs are $|1\rangle$.

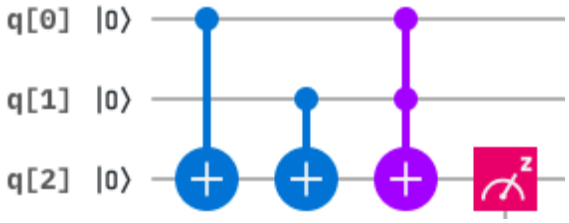

Fig. 4 Circuit for OR operation. The third qubit is the output qubit, resulting in eigenstate $|1\rangle$ when either of the inputs is $|1\rangle$.

Fig. 5 Circuit for XOR operation. The third qubit is the output qubit, resulting in eigenstate $|1\rangle$ when even number of the inputs are $|1\rangle$.

representing the requisite functions. This flexibility allows complex behaviour to be added to the quantum robot.

4 Demonstration of adding a degree of freedom

Quantum Braitenberg vehicles have the ability to utilise truly quantum phenomena like superposition and/or entanglement to achieve a non-deterministic, probabilistic relationship between the inputs and outputs. In this section, we develop techniques to derive circuits from arbitrary desired input-output relationships. We remind the reader that all inputs and outputs are either $|0\rangle$ or $|1\rangle$ based on the absence or the presence of some signal, respectively. In this section, we first demonstrate reducing a non-deterministic input-output relation as given in table 1, and then propose schemes to efficiently realise

Table 1 Truth table for a quantum Braitenberg vehicle with three degrees of movement. $S1$, $S2$, $S3$ are the three inputs, while $M1$, $M2$, $M3$ are the three outputs. The outputs are fed to the circuit as inputs in the next iteration of the robot.

S1	S2	S3	M1	M2	M3
0	0	0	0	0	1
0	0	1	010 or 011		
0	1	0	0	1	1
0	1	1	100 or 101		
1	0	0	1	0	1
1	0	1	110 or 111		
1	1	0	1	1	1
1	1	1	000 or 001		

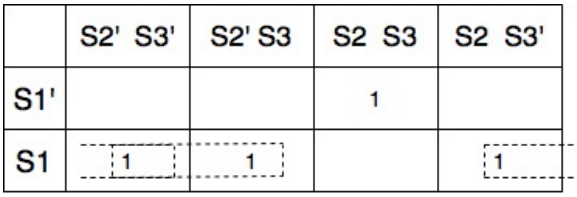

Fig. 6 K-map representation of $M1$. The reduced expression for $M3$ is $S1'.S2.S3 + S1.S3' + S1.S2'$, where $+$ represents OR, $.$ represents AND, and $'$ represents NOT.

such reductions. We also give reduction of four-degree non-deterministic input-output relations in the Appendix (section 6). Generalisation to higher degrees of freedom is straight-forward, though we note that more efficient reduction techniques than K-maps, like Quine-McCluskey method need to be used, or reductions based on Boolean algebra laws need to be adopted.

4.1 Reduction of non-deterministic functions

Consider the following desired non-deterministic behaviour for the robot:

The K-map representations of $M1$, $M2$, and $M3$ are as follows. The map in Fig. 6 use standard K-map properties, while map 7 introduces the notion of non-deterministic K-map: maps containing non-deterministic outputs.

Similarly, K-map representation for $M2$ can be constructed. The reduced expression for $M2$ will be $S2'.S3 + S2.S3'$. Note from table 1 that in all of input states for which outputs are non-deterministic, non-determinism is caused by output $M3$. We thus represent this non-determinism in form of a non-deterministic K-map as in Fig. 7.

Reduction occurs using the same techniques as those of standard K-maps. $M3$ is therefore reduced to $S3' + H(S3)$, where $H(.)$ represents the input combination for which $M3$ is non-deterministic. This is interpreted as $M3 = 1$ (deterministic) whenever $S3' = 1$ and $M3 = 0/1$ (non-deterministic) whenever $S3 = 1$.

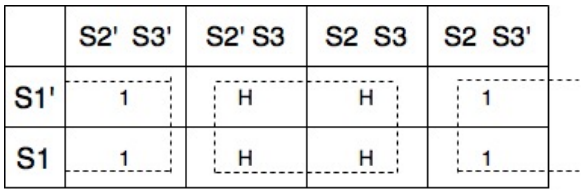

Fig. 7 K-map representation of $M3$, where $+$ represents OR, . represents AND, and $'$ represents NOT.

4.2 Implementation techniques

Standard K-map (or other Boolean expression reductions) techniques aim to reduce a provided expression into either the SOP (sum of products) form or the POS (product of sums) form, with unique interconversions between the two. Arbitrary SOP expressions require AND operations on a series of inputs of lengths ≥ 1 and OR operations on the results of these AND operations, while arbitrary POS expressions require OR operations on a series of inputs of lengths ≥ 1 and AND operations on the results of these OR operations. It is thus essential to develop techniques to perform AND and OR operations on arbitrary number of qubits. We introduce two techniques to perform AND on arbitrary number of qubits. OR can be similarly constructed.

Consider $AND(q_1, q_2, q_3, \dots, q_n)$ where n is odd (result for even n follows) and q_i represents the ith qubit. Then $AND(q_1, q_2, ..., q_{n-1})$ contains AND on a set of even qubits such that $AND(q_1, q_2, q_3, \dots, q_n) = AND(q_1, q_2, ..., q_{n-1})$ AND q_n. We pair these qubits and perform the operation outlined in Fig. 3: $ccx(q_1, q_2, q_1^{'})$, $ccx(q_3, q_4, q_2^{'})$, up to $ccx(q_{n-2}, q_{n-1}, q_{\frac{n-1}{2}}^{'})$, where $ccx(a, b, c)$ implies performing the operation in Fig. 3 on input qubits a and b and storing the operation result in qubit c. We now get $AND(q_1, q_2, q_3, \dots, q_n) = q_n$ AND $AND(q_1^{'}, q_2^{'}, ..., q_{\frac{n-1}{2}}^{'})$. Recursively, we can solve for the desired AND on the initial n qubits. We note that had n been initially even, we would not have to remove the last qubit in the first step.

This construction reduces the size of the input qubit list by half in each call, accumulating to approximately $\mathcal{O}(log\, n)$ calls to the recursive AND routine. Moreover, one $ccx(a, b, c)$ operation takes constant time, and there are $\mathcal{O}(n)$ such operations for input size n. The main disadvantage is the number of auxiliary qubits required: $\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + ... + 1$ (convergent finite G.P.).

We now propose an improvement. We note that after $ccx(q_i, q_j, q_k)$, q_k is either in $|0\rangle$ or in $|1\rangle$ because a single AND operation on two qubits is deterministic as defined in section 3. We also have a sufficiently large classical register that stores the measurement results; such a register is initialised at the time of constructing the quantum circuit (that combines a quantum register and a classical register). We introduce an operation $ccxM(q_i, q_j, q_k, c_p)$ which involves $ccx(q_i, q_j, q_k)$ followed by measurement of q_k with the result stored in c_p, or pth position in the classical register. Our initial $AND(q_1, q_2, q_3, \dots, q_n)$ for even n can be broken down as a series of operations: $ccxM(q_1, q_2, q_{n+1}, c_1)$,

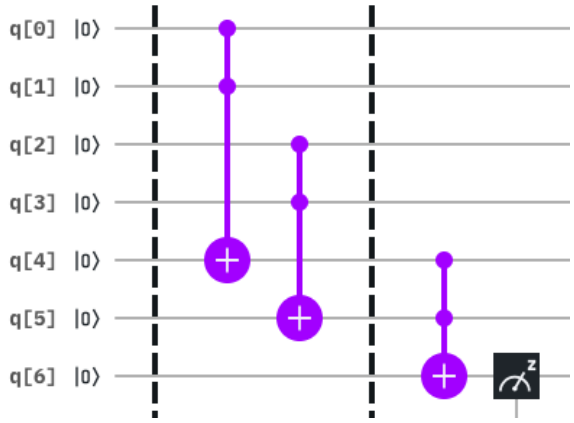

Fig. 8 A sample circuit with $AND(q[0], q[1], q[2], q[3])$ using the initial recursive AND routine method.

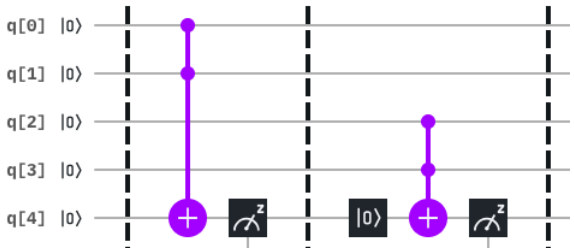

Fig. 9 A sample circuit demonstrating the improved implementation technique for $AND(q[0], q[1], q[2], q[3])$. The auxiliary qubit needs to be reset back to $|0\rangle$ to avoid *carrying on* the result of the operation previous to it.

followed by $ccxM(q_3, q_4, q_{n+1}, c_2)$, up to $ccxM(q_{n-1}, q_n, q_{n+1}, c_{\frac{n}{2}})$. The final result is obtained by $\frac{n}{2}$ bitwise-AND operations from c_1 to $c_{\frac{n}{2}}$. In the case of odd n, the expression breaks down as stated earlier: $AND(q_1, ..., q_{n-1})\, AND\, q_n$ followed by $ccxM(q_1, q_2, q_{n+1}, c_1)$ up to $ccxM(q_{n-2}, q_{n-1}, q_{n+1}, c_{\frac{n}{2}})$ followed by a simple $CNOT$ operation $cxM(q_n, q_{n+1}, c_{\frac{n}{2}+1})$. Performing bitwise-$AND$ from c_1 to $c_{\frac{n}{2}+1}$ produces the desired result.

We note that the main advantage of this improved version is the reduction in the number of auxiliary qubits required from the previous version. We also observe from Fig. 9 and from the discussion on $ccxM$ that irrespective of the number of qubits to operate upon (i.e. the value of n), we measure the auxiliary qubit immediately after the result of the operation is stored in it. Less number of auxiliary qubits, short time interval between the operation and the measurement, and the surety that the auxiliary qubit is either $|0\rangle$ or $|1\rangle$ (since AND operation is deterministic) with probability arbitrarily close to 1 (accounting for arbitrarily small gate application errors) ensure almost negligible decoherence possibilities and thus error-free operation of the circuitry.

4.3 Sample implementation

Here, we provide a sample implementation of the functions reduced in section 4.1. We provide an algorithmic view to realise the techniques discussed. This can be converted to practical implementations using any programming language supporting quantum programming. The authors use *Qiskit* in *Python*.

Algorithm 1: twoQubitAND(qubit1, qubit2, qubit1NOT, qubit2NOT, cPos)

Result: Performs AND operation on qubits qubit1 and qubit2 and stores the measurement in position cPos in the classical register. Binary variables qubit1NOT and qubit2NOT denote whether qubit1 and qubit2 respectively must be flipped before the AND operation.

```
if qubit1NOT then
  | circuit.x(quantumRegister[qubit1])
end
if qubit2NOT then
  | circuit.x(quantumRegister[qubit2])
end
circuit.ccx(quantumRegister[qubit1], quantumRegister[qubit2],
  quantumRegister[auxiliaryQubit])
circuit.measure(quantumRegister[auxiliaryQubit], classicalRegister[cPos])
if qubit1NOT then
  | circuit.x(quantumRegister[qubit1])
end
if qubit2NOT then
  | circuit.x(quantumRegister[qubit2])
end
circuit.reset(quantumRegister[auxiliaryQubit])
```

From algorithm 2, we note that a simple loop performing bitwise-AND over the elements of the *classicalRegister* returns the output of performing the AND operation over the given list of n qubits. We now present a way to perform $H(.)$ discussed in Fig. 7, in algorithm 3, while the method to build and run the circuit in algorithm 4.

In all the algorithms 1, 2, and 3, *circuit.x()*, *circuit.h()*, *circuit.reset()*, *circuit.cx()*, *circuit.ccx()*, and *circuit.measure()* represent the usual quantum operations X, H, $|0\rangle$ reset, $CNOT$, *Toffoli*, and measurement respectively. The algorithm 4 builds the requisite circuit for the truth table given in 1. For instance, the term $S1'.S2.S3'$ translates to the n qubit AND operation $nQubitAND([0, 1, 2], [1, 0, 1])$ where $S1$, $S2$, and $S3$ are represented by 0, 1, and 2 respectively, whereas the second list has 1 for the qubits that need to be flipped before performing the AND operation (in this case, $S1'$ and $S3'$ are represented by 1's in their respective positions 0 and 2). The $nQubitAND$ algorithm thus inserts a X gate before the AND operation and measurement, and then removes this X gate to revert qubits $S1$ and $S3$ to their initial values ready for the next operation.

Algorithm 2: nQubitAND(qubitList, qubitNOTList)

```
Result: Performs AND operation n qubits given by the list qubitList.
        qubitNOTList is a list of n binary values depicting whether each of the
        qubits in qubitList need to be flipped before AND is performed on them.
classicalRegisterIndex = 0
length = length(qubitList)
// handling odd number of qubits
if length % 2 != 0 then
    if qubitNOTList[length - 1] == 1 then
        circuit.x(quantumRegister[qubitList[length - 1]])
    end
    // cx operation on the last qubit as discussed before
    circuit.cx(quantumRegister[qubitList[length - 1]],
      quantumRegister[auxiliaryQubit])
    circuit.measure(quantumRegister[auxiliaryQubit],
      classicalRegister[classicalRegisterIndex])
    circuit.reset(quantumRegister[auxiliaryQubit])
    classicalRegisterIndex = classicalRegisterIndex + 1
    if qubitNOTList[length - 1] == 1 then
        circuit.x(quantumRegister[qubitList[length - 1]])
    end
    length = length - 1
end
i = length - 1
while i ≥ 0 do
    twoQubitAND(qubitList[i], qubitList[i - 1], qubitNOTList[i], qubitNOTList[i -
      1], classicalRegisterIndex)
    classicalRegisterIndex = classicalRegisterIndex + 1
    i = i - 2
end
```

Algorithm 3: nQubitHadamard(qubitList, qubitNOTList)

```
Result: perform H(qubitList) or Hadamard operation on the auxiliary qubit if
        AND(qubitList) returns 1
output = nQubitAND(qubitList, qubitNOTList)
if output == 1 then
    circuit.h(quantumRegister[auxiliaryQubit])
    circuit.measure(quantumRegister[auxiliaryQubit], classicalRegister[0])
    circuit.reset(quantumRegister[auxiliaryQubit])
    // execute the circuit
    return classicalRegister[0]
end
return 0
```

We compare two runs of the above construction, beginning with the state 000. Run 1: 000 → 001 → 010 → 011 → 101 → 111 → 001 → 011 → 100 → 101 → 110 while run 2: 000 → 001 → **011** → **100** → 101 → 111 → 001 → 011 → **101** → **110** → **111**. The bolded states are the differences between run 1 and run 2, establishing truly non-deterministic nature of the construction, and verifying the implementation of the desired input-output relationship.

Algorithm 4: robotOperation()

Result: Builds the requisite circuit and iterates to get the requisite input-output relation.

M1 = $nQubitAND([0, 1, 2], [1, 0, 0])$ OR $nQubitAND([0, 2], [0, 1])$ OR $nQubitAND([0, 1], [0, 1])$
M2 = $nQubitAND([1, 2], [1, 0])$ OR $nQubitAND([1, 2], [0, 1])$
M3 = $nQubitAND([2], [1])$ OR $nQubitHadamard([2], [0])$
// insert these outputs as inputs to the circuit and repeat
if $M1 == 1$ **then**
    // force S1 to be $|1\rangle$
    circuit.reset(quantumRegister[0])
    circuit.x(quantumRegister[0])
end
if $M2 == 1$ **then**
    // force S2 to be $|1\rangle$
    circuit.reset(quantumRegister[1])
    circuit.x(quantumRegister[1])
end
if $M3 == 1$ **then**
    // force S3 to be $|1\rangle$
    circuit.reset(quantumRegister[2])
    circuit.x(quantumRegister[2])
end
// repeat for desired number of steps

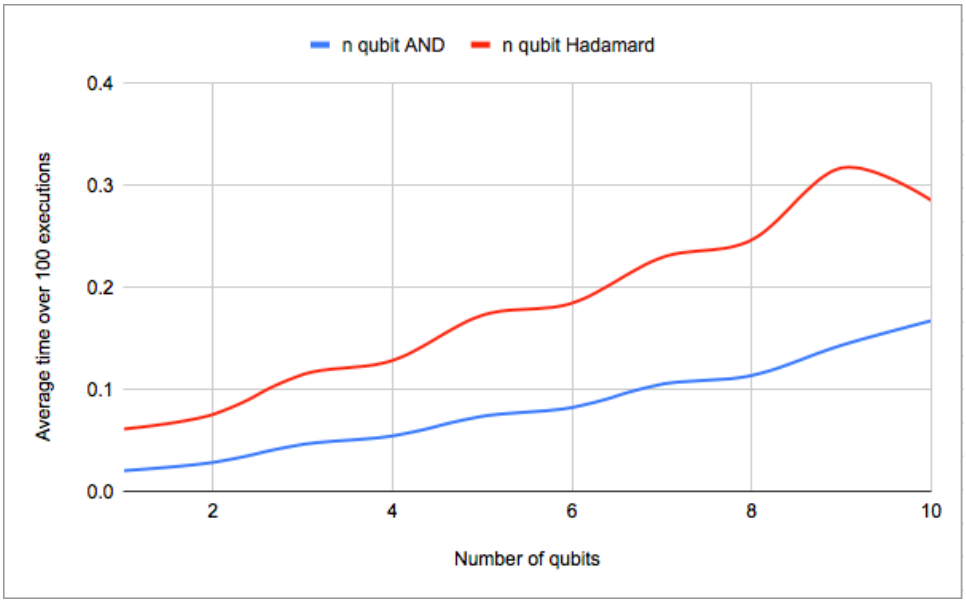

Fig. 10 Running time in **seconds** for various values of n. Simulations were run using *Qiskit Aer* on Tesla K80 GPU hosted on Google Colab. We conclude the implementation is fast enough for implementation of the complete circuitry of a quantum Braitenberg vehicle. See Appendix for details.

5 Conclusion

To conclude, we have established here the possibility to add the desired number of degrees of freedom to a quantum Braitenberg vehicle since the logic used in such robots can be implemented in the same way as classical Boolean logic is implemented. This is possible due to co-measurability of the propositions involved, as discussed in Section 3. We then established quantum implementation of basic classical Boolean logic gates and reasoned that any logic function can be implemented in a quantum robot. We then introduced ideas in subsections 4.1 and 4.2, that in addition to probably being of independent interest to the reader, allow for efficient implementation of the desired input-output behaviour.

The popularity of classical Braitenberg vehicles lies in the flexibility of implementable Boolean functions allowing complex behaviours in a robot. A quantum Braitenberg vehicle can, in addition, take advantage of quantum phenomena to exhibit non-deterministic nature. It also has the flexibility to accommodate more degrees of freedom, thus allowing the possibility of complex functions impossible for a classical Braitenberg vehicle, leading to several possible and exciting applications such as simulation of non-Markovian decision processes or studies on non-supervised learning for non-deterministic agents. Robotics has been the center of academic and attraction since its inception. In the nascent field of quantum robotics, the ability to harness powers of quantum mechanics provides avenues yet to be explored. As quantum architecture increases in efficiency thereby increasing coherence time and the rate of errors in application of quantum gates decreases, we are set to witness astonishing progress in the field of quantum robotics.

Acknowledgments

N.M. and S.C.R. would like to thank IISER Kolkata for providing hospitality during which this work has been done. B.K.B. acknowledges the support of IISER-K Institute fellowship. The authors acknowledge the support of IBM Quantum Experience and Qiskit for producing experimental results. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM quantum experience team. The authors are grateful to the anonymous reviewers for their insightful comments that led to improvements in the text.

References

1. P. Benioff, The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines, J. Stat. Phys. **22**, 563 (1980).
2. R. Landauer, The physical nature of information, Phys. Lett. A **217**, 188 (1996).
3. R. Feynman, Simulating physics with computers, Int. J. Theor. Phys. **21**, 467 (1982).
4. Y. Manin, Computable and uncomputable, Sovetskoye Radio, Moscow, (1980).

5. D. Deutsch, Quantum theory, the Church-Turing principle and the universal quantum computer, Proceedings of the Royal Society, (1985).
6. P. Benioff, Quantum robots and environments, Phys. Rev. A **58**, 893 (1998).
7. G. Weihs, T. Jennewein, C. Simon, H. Weinfurter, and A. Zeilinger, Violation of Bell's Inequality under Strict Einstein Locality Conditions, Phys. Rev. Lett. **81**, 5039 (1998).
8. D. Y. Dong, C. L. Chen, C. B. Zhang, and Z. H. Chen, Quantum Mechanics Helps in Learning for More Intelligent Robots, Chin. Phys. Lett. **23**, 1691 (2006).
9. D. Y. Dong, C. L. Chen, C. B. Zhang, and Z. H. Chen, Quantum robot: structure, algorithms and applications, Robotica, **24**, 513 (2006).
10. Z. Toffano and F. Dubois, Eigenlogic: Interpretable Quantum Observables with applications to Fuzzy Behavior of Vehicular Robots, arXiv:1707.05654.
11. P. A. Zizzi, I, Quantum Robot: Quantum Mind control on a Quantum Computer, arXiv:0812.4614.
12. S. Mahanti, S. Das, B. K. Behera, and P. K. Panigrahi, Quantum Robots Can Fly; Play Games: An IBM Quantum Experience, Quantum Inf. Process. **18**, 219 (2019).
13. N. Mishra, R. C. Sarath, B. K. Behera, and P. K. Panigrahi, Automation of Quantum Braitenberg Vehicles Using Finite Automata: Moore Machines. DOI: 10.13140/RG.2.2.12300.16003.
14. V. Braitenberg, Vehicles: Experiments in Synthetic Psychology, MIT Press; Reprint edition (1986).
15. A. Raghuvanshi, Y. Fan, M. Woyke, and M. Perkowski, Quantum Robots for Teenagers, Proceedings of The International Symposium on Multiple-Valued Logic (2007).
16. G. Birkhoff and J. V. Neumann, The Logic of Quantum Mechanics. Annals of Mathematics Second Series, Vol. 37, No. 4 (Oct., 1936), pp. 823-843

6 Appendix I

Consider a robot to have the desired behaviour as in table 2.

Table 2 Truth table for a quantum Braitenberg vehicle with four degrees of movement. A, B, C, and D are the inputs, while P, Q, R, and S are the outputs. The outputs are fed to the circuit as inputs in the next iteration of the robot.

A	**B**	**C**	**D**	**P**	**Q**	**R**	**S**
0	0	0	0	1100 or 1101 or 1110 or 1111			
0	0	0	1	0	1	0	1
0	0	1	0	0010 or 0011 or 0000 or 0001			
0	0	1	1	0101 or 0111 or 1101 or 1111			
0	1	0	0	1	0	0	0
0	1	0	1	1000 or 1010 or 1100 or 1110			
0	1	1	0	1001 or 1011 or 1101 or 1111			
0	1	1	1	1	0	1	1
1	0	0	0	0000 or 0001 or 1000 or 1001			
1	0	0	1	0010 or 0011 or 0110 or 0111			
1	0	1	0	1	1	1	0
1	0	1	1	0011 or 0111 or 1011 or 1111			
1	1	0	0	0100 or 0110 or 1100 or 1110			
1	1	0	1	0	0	0	1
1	1	1	0	0100 or 0101 or 0110 or 0111			
1	1	1	1	0001 or 0011 or 0101 or 0111			

	C' D'	C' D	C D	C D'
A' B'	1		H	
A' B	1	1	1	1
A B	H			
A B'	H		H	1

Fig. 11 K-map representation of P. The reduced expression for P is $A.B'.C.D' + A'.B + A'.C'.D' + H(A.C'.D') + H(B'.C.D)$, where $+$ represents OR, $.$ represents AND, and $'$ represents NOT.

Similar reductions for Q, R, and S yield: $Q = A.C.D' + A.B.D' + H(A.C.D) + H(A.B'.D) + A'.B'.D + A'.B'.C' + H(A'.B.C.D') + H(A'.B.C'.D)$, $R = A.B'.C + A.B'.D + H(A.B.C) + H(A.B.D') + H(A'.C.D') + H(A'.B'.D') + H(A'.B'.C) + H(A'.B.C'.D) + A'.B.C.D$, and $S = CD + A'.B.C + A'.B'.D + A.B.D + H(A.B'.C') + H(A'.B'.D') + H(A.B.C.D')$. We run algorithm 4 on these expressions and obtain the following runs, starting from state 0000.

$Run\,1$: 0000 $\rightarrow$ 1110 $\rightarrow$ 0111 $\rightarrow$ 1011 $\rightarrow$ 1111 $\rightarrow$ 0101 $\rightarrow$ 1110 $\rightarrow$ 0111 $\rightarrow$ 1011 $\rightarrow$ 1011 $\rightarrow$ 0111

$Run\,2$: 0000 $\rightarrow$ **1111** $\rightarrow$ **0011** $\rightarrow$ **0101** $\rightarrow$ **1110** $\rightarrow$ **0111** $\rightarrow$ **1011** $\rightarrow$ **1111** $\rightarrow$ **0101** $\rightarrow$ **1100** $\rightarrow$ **1110**

It is clearly observed the differences between the two runs. Run 1 takes nearly 24.88349962234497 seconds while run 2 takes nearly 24.779985666275024 seconds using *Qiskit Aer* quantum simulator on Tesla K80 GPU hosted on Google Colab. We note there are 29 operations combined for the four input states for one step and we run the simulation for ten steps, which lead to an average of nearly 0.08580 seconds and 0.8544822 seconds per operation in the respective runs. We also note that 21 of the 29 operations involve at least a three qubit AND, which in turn involves a two qubit AND (which has at most two X gates, one *Toffoli* gate, one measurement gate, and one reset gate) and a $cxM()$ operation (at most one X, one $CNOT$, one measurement gate, and one reset gate), amounting to a total of at most nine gates per three qubit AND operation. The average time per operation (approximately 0.08 seconds) coincides with the results in Fig. 10. It is to be noted that for simulations run on local systems, the time reported is incorrect in the sense that the process running the simulation is interrupted by the operating system to give control of the CPU to other processes, and this *waiting/idle* time is added in the reported time value. The actual running time is the reported value minus the time spent by the process waiting in the ready queue of the CPU. Analytically, we have a sufficiently efficient implementation. Further programming

optimisations are possible: caching the circuits in real-time or catching outputs (similar to techniques in dynamic programming problems).